Examining Semantic Diversity and Semantic Locality of Operators in Genetic Programming

Quang Uy Nguyen,
M.Sc. Vietnamese Military Technical Academy

The thesis is submitted to University College Dublin
for the degree of Ph.D.
at the School of Computer Science and Informatics

Research Supervisors
Dr. Michael O'Neill
Dr. Xuan Hoai Nguyen

July 18, 2011

Abstract

Diversity, the ability of a searcher to explore different parts of the search space, and locality, the ability of a searcher to exploit a specific area of the search space, have long been seen as crucial properties for the efficiency of Evolutionary Algorithms in general, and Genetic Programming (GP) in particular. A number of studies investigating the effects of diversity and locality in GP can be found in the literature. However, most previous work on diversity, and all on locality, focus solely on syntactic aspects; semantic diversity and locality of operators have not been thoroughly investigated. This thesis investigates the role of semantic diversity and semantic locality of operators in GP.

This thesis proposes a novel way to measure semantics in GP by sampling a number of points from the problem domain. This semantics is called *Sampling Semantics*. From that, a semantic distance and two semantic relationships between subtrees are defined. Based on these metrics, a number of novel semantic based genetic operators (crossovers and mutations) are introduced. These operators address two main objectives: Promoting semantic diversity and improving semantic locality. The new semantic based crossovers and mutations are tested on a number of real valued symbolic regression problems and the experimental results show the positive impact of promoting semantic diversity and the greater improvement of enhancing semantic locality.

Since crossover has long been seen as the primary operator in GP, the thesis places an emphasis on studying semantic based crossovers. These semantic based crossovers are analysed on some important properties of GP. The results show that semantic based crossovers achieve greater semantic diversity and higher semantic locality that leads to more constructive effect (more frequently generate children that are better then their parents) in comparison with standard crossover. This analysis shed some light on the improved performance of semantic based crossovers.

Furthermore, a deep analysis of the behaviour of semantic based crossovers are investigated. Aspects under investigation include the generalisation ability of semantic based crossover, the comparison between semantic locality and syntactic locality, the ability of

semantic based crossovers to deal with increasingly difficult problems and their impact on the fitness landscape. The experimental results show that the generalisation ability of semantic based crossovers is better than standard crossover, that semantic locality is more important than syntactic locality in improving GP performance, and the ability of GP to generalise. They also show that semantic based crossovers deal well with increasingly difficult problems and that improving semantic locality helps to smooth out the fitness landscape of a problem.

Finally, the idea of promoting semantic diversity and enhancing semantic locality are extended to the Boolean domain. For Boolean problems, new semantic based crossovers are proposed. These crossovers are then tested on some well-known Boolean problems and the results again show that promoting semantic diversity is important with Boolean problems and that improving semantic locality even leads to a further improvement of GP performance.

In summary, this thesis highlights the important role semantics has to play in managing diversity and locality in GP.

Acknowledgments

The first person I would like to thank is my supervisor, Dr Michael O'Neill, the director of Natural Computing Research and Applications Group, University College Dublin (UCD NCRA), for guiding me through the PhD process. Michael's enthusiasm and genius has always been a valuable source of help. His encouragement has inspired much of the research in this thesis.

I also wish to thank my co-supervior, Dr Xuan Hoai Nguyen, the director of Information Technology Centre, Hanoi University, Vietnam. Although living far away, we had an online research chat almost every week. Working with Hoai, I have learnt how to love and how to do research. Hoai also shared his experiences of moving abroad and living in a new country and new society.

Professor Bob McKay at the Structural Complexity Laboratory, Seoul National University, Korea has discussed a lot of issues relating to this topic with me. I would like to thank him for his thorough comments and suggestions on my research.

All the member of UCD NCRA have helped me during my PhD research. I would like to thank all of them. In particular, I wish to express my gratitude to Dr Erik Hemberg for taking time to proof read this thesis.

This research was funded under a Postgraduate Scholarship from the Irish Research Council for Science Engineering and Technology (IRCSET). Without this funding, I could not have had a chance to live and do research in a country renowned as Ireland. I would like to express my thank to IRCSET for their invaluable funding.

Last, but most important, I would like to dedicate this work to my family, to my parents, Nguyen Van Hien and Nguyen Thi May, for working so hard to bring up their son, to my wife, Nguyen Thi Minh Hang for sharing a lot of happiness and difficulty in the life with me and to my daughter, Nguyen Bao Ngoc for trying to grow up without her dad taking care for the past three years.

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Meaning
GP	Genetic Programming
SS	Sampling Semantics
SSD	Sampling Semantic Distance
SC	Standard Crossover
SAC	Semantic Aware Crossover
SSC	Semantic Similarity based Crossover
SAM	Semantic Aware Mutation
SSM	Semantic Similarity based Mutation
LBSS	The lower bound of semantic sensitivity
UBSS	The upper bound of semantic sensitivity
AGP	Attributes Genetic Programming
SASE	Self-Adaptive Successful Execution
MSSC	The Most Semantic Similarity based Crossover
NSM	No Same Mate Selection
SBS	Soft Brood Selection
SySC	Syntactic Similarity based Crossover
VAL	Validation Set Method
MUL	Multi-objective Method
TBC	Tarpeian Bloat Control
GCSC	Guaranteed Change Semantic Crossover
LCSC	Locality Controlled Semantic Crossover
SDC	Semantic Driven Crossover

Publications

[1] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, R. I. McKay, and Edgar Galvan-Lopez. Semantically-based crossover in genetic programming: application to real valued symbolic regression. *Genetic Programming and Evolvable Machines*, 2011. Vol: 12(2), Pages: 91-119, ISSN: 1389-2576.

[2] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Examining the landscape of semantic similarity based mutation. *In The Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, Dublin, Ireland, 12-16 July 2011. ACM.

[3] Quang Uy Nguyen, Thi Hien Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Improving the generalisation ability of genetic programming with semantic similarity based crossover. *In The Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of LNCS, Istanbul, 7-9 April 2010. Springer.

[4] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, and Bob Mckay. Semantics based crossover for boolean problems. *In Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO 2010, Portland, Oregon, 7-11 July 2010. ACM.

[5] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, and Bob McKay. The role of syntactic and semantic locality of crossover in genetic programming. *In The Proceedings of 11th International Conference on Parallel Problem Solving From Nature, PPSN 2010*, volume 6239 of Lecture Notes in Computer Science, pages 533-542, Krakow, Poland, 11-15 September 2010. Springer.

[6] Quang Uy Nguyen, Bob McKay, Michael O'Neill, and Xuan Hoai Nguyen. Self-adapting semantic sensitivities for semantic similarity based crossover. *In The Proceedings of IEEE Congress on Evolutionary Computation, CEC 2010*, Barcelona, Spain, 7-11 July 2010. IEEE Press.

[7] Quang Uy Nguyen, Michael O'Neill, and Xuan Hoai Nguyen. Predicting the tide with genetic programming and semantic based crossovers. *In The Proceedings of The Second International Conference on Knowledge and Systems Engineering*, Hanoi, Vietname, 7-9 October 2010. IEEE Computer Society, IEEE Press.

[8] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael. O'Neill. Semantic aware crossover for genetic programming: the case for real valued function regression. *In The Proceedings of the 12th European Conference on Genetic Programming, EuroGP09*, pages 292-302. Springer, April 2009.

[9] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Semantics based mutation in genetic programming: The case for real valued symbolic regression. *In The Proceedings of 15th International Conference on Soft Computing, Mendel09*, pages 73-91, Brno, Czech Republic, June 24-26 2009.

[10] Quang Uy Nguyen, Michael O'Neill, Xuan Hoai Nguyen, Bob McKay, and Edgar Galvan Lopez. Semantic similarity based crossover in GP: The case for real valued function regression. *In The Proceedings of Evolution Artificial, 9th International Conference, EA 2009*, Lecture Notes in Computer Science, pages 13-24, October 2009.

[11] Quang Uy Nguyen, Michael O'Neill, Xuan Hoai Nguyen, Bob McKay, and Edgar Galvan Lopez. An analysis of semantic aware crossover. *In The Proceedings of the International Symposium on Intelligent Computation and Applications, ISICA 2009.* Springer-Verlag, 2009.

Chapter 1

Introduction

Evolutionary Algorithms are a class of population based algorithms inspired by Darwin's theory of evolution. One of the subsets of evolutionary algorithms is Genetic Programming – a mechanism designed to allow a population of computer programs to be evolved. In Genetic Programming, three main components that influence its performance are its representation, operators and a method to determine the ability to solve the problem of each program (a fitness function). Among them, the operators drive the movement of the programs in the search space. Since the design of previous operators often lacks semantic information, it affects the performance of Genetic Programming and limits its ability to solve problems. This thesis aims to promote the semantics of the operators in Genetic Programming and examines its impact on the behaviour of Genetic Programming search.

1.1 Motivation

Genetic Programming (GP) is a biologically inspired method of using a computer to evolve solutions, in the form of computer programs, for a problem [98, 154]. To solve a problem using a GP system, a population of indiduals is first initialised. Each individual, often represented in the form of a tree, is a solution to the problem. The population is then evolved, under fitness based selection, through a number of generations by applying genetic

operators such as crossover and mutation, where crossover is considered as the primary operator. The evolutionary process terminates when a desired solution is found or when the maximum number of generations is exceeded.

Since its introduction, GP has been applied to a large number of fields [154, 100]. So far, GP research has focused on syntactic aspects of GP representation, which has brought valuable insights and contributions to the sucess of GP [98, 156, 108, 150, 79]. However, from a normal programmer's perspective, maintaining syntactic correctness is only one part of program construction: programs must not only be syntactically correct, but also semantically correct. Thus incorporating semantic awareness in the GP evolutionary process could potentially improve its performance and extend the applicability of GP to problems that are difficult to deal with by using purely syntactic approaches.

In the field of Natural Language Processing, Psychology and Computer Science, semantics has been extensively studied [2, 27, 142]. While the definition of semantics changes from field to field, in Computer Science, semantics is often considered as the application of mathematical logic or formal semantics of programming languages. In this field, the semantics of a program or a function reflects the meaning of that program or that function. As the main aim of GP is to evolve computer programs, it is very interesting and important to take the idea of semantics from other fields of Computer Science into the GP evolutionary process.

The idea of incorporating semantics into GP evolutionary process is not entirely new. Some research has been done in this field recently [82, 83, 11, 13, 123, 103]. However, an extensive investigation the impact of both semantic diversity and semantic locality of operators in GP has still not been undertaken. This research is one of the first attempts to thoroughly examine the effect of semantic diversity and semantic locality of operators on a number of aspects of GP.

1.2 Research Aims

The objective of this thesis is to examine the important role of semantic diversity and semantic locality of operators in GP. In particular, we address the following questions.

1. *How to measure semantics, especially for real valued problems, in GP?*
2. *How to design semantic based operators that promote semantic diversity and semantic locality?*
3. *How do semantic based crossovers effect some important properties (diversity, locality, constructive effect, code bloat, etc.) of GP?*
4. *Do semantic based crossovers improve the ability of GP to generalise?*
5. *Whether syntactic locality or semantic locality is more important?*
6. *How do semantic based crossovers deal with increasingly difficult problems?*
7. *Does improving semantic locality of crossover help to smooth out a fitness landscape?*

1.3 Contributions

The investigation of semantic diversity and semantic locality of operators in GP has given rise to a number of contributions which are outlined as follows:

A literature review: This thesis presents a thorough review of previous studies related to the research in the thesis. They include a survey of previous research on using semantics in GP, alternative operators (crossovers and mutations), and the previous studies of diversity and locality in GP.

A novel way to measure semantics in GP: The thesis proposes a novel way to measure semantics of an individual or a subindividual (subtree) in GP. The semantics can be determined by sampling a number of points from the problem domain. From that a

semantic distance and two semantic relationships are defined. These semantic relationships facilitate the design new semantic based operators.

New semantic based operators: The thesis also proposes a number of novel semantic based operators. These operators address two main objectives: promoting semantic diversity and improving semantic locality. The new operators are tested on a number of problems and the results show that they are comparative in comparison with the standard operators and some similar fitness based operators. Moreover, the superiority of the new semantic based operators is not only on training data, but also on testing data, and not only on real valued problems but also on Boolean problems.

New GP representation: A new GP tree-based representation is also introduced. The new tree-based representation is formed by adding a number of attributes to every node in the traditional tree-based representation. These attributes are used to store semantics of each subtrees in a GP individual. This attributes-based representation help to speed up the semantic checking process and guarantee for executing some semantic-based operators.

A study of the behaviour of GP under semantic based crossovers: The new semantic based operators are also used to investigate a number aspects of GP. These aspects include semantic diversity and semantic locality of an operator, the constructive effect of operators, code bloat effect, the ability of GP to generalise, and the impact of operators to fitness landscape. These examinations shed light on the superior performance of the new semantic based crossovers.

1.4 Scope Limitations

Although, semantics has been broadly used in the field of Natural Language Processing, Psychology and Computer Science [2, 27, 142], this thesis focuses on studying semantics in the context of GP. There are many good studies regarding semantics in other fields, which are beyond the scope of this thesis, can be found in the literature.

When running an evolutionary algorithm, there are a number of parameters (initialisation method, crossover rate, mutation rate, etc.) which need to be considered. This thesis

in no way can exhaustively study different settings for these parameters. Therefore, typical settings that have been used in the previous research [98, 67] are used for the experiments in the thesis.

Over two decades of development, GP has been successfully applied to a wave of problems [154, 100]. This thesis studies semantic based operators on two popular benchmark problems. These problems include real valued symbolic regression problems and Boolean problems. Both of them have been widely used for the experiments in GP [98, 67, 11, 13].

1.5 Thesis Overview

The remainder of this thesis is organised as follows. In Chapter 2, we give a brief introduction to Evolutionary Algorithms and a more detailed introduction to Genetic Programming.

Chapter 3 gives a review of related work to the research in the thesis. They include a survey of different ways of using semantics in GP, alternative crossovers and mutations, and the previous studies of diversity and locality in GP. This chapter is intended to provide a general understanding of the history of these problems and also indicates that semantic diversity and semantic locality of an operator in GP are still a rather new area.

Chapter 4 proposes a novel way to measure semantics in GP. The semantics of any subtree (tree) in GP is qualified by sampling a number of points from the problem domain. A semantic distance and two semantic relationships are defined from the qualified semantics. Following this, several semantic based crossovers and mutations are introduced. These operators focus on promoting semantic diversity and improving semantic locality of GP. This chapter serves as the foundation for the thesis, giving new semantic based operators that will be further investigated in the following chapters.

The rest of the thesis (excluding the last chapter: Conclusions and Future Work) can be divided into three parts. The first part includes two chapters (Chapter 5 and Chapter 6), which examine the performance of semantic based operators compared to standard operators and several related fitness based methods. The second part, consisting of four chapters (Chapter 7 to Chapter 10), concentrates on analysing the behaviour of

GP under the impact of semantic based crossovers. The last part contains two chapters (Chapter 11 Chapter 12) intending to extend the application of semantic based crossovers to other problem domains in addition to real-value symbolic regression problems. Each chapter is discussed in detail in the following paragraphs.

In Chapter 5 we compare the performance of semantic based operators with standard operators. This comparison includes contrasting Semantic Aware Crossover (SAC), Semantic Similarity based Crossover (SSC) with standard crossover and Semantic Aware Mutation (SAM), Semantic Similarity based Mutation (SSM) with standard mutation. A combination of both crossover and mutation for improving semantic locality, SSC and SSM, is also investigated. Finally, the impact of some parameters on Semantic Similarity based Crossover and on Semantic Similarity based Mutation is analysed. Since, crossover has widely been seen as the primary operator in GP, the succeeding chapters only focus on studying semantic based crossovers.

Chapter 6 proposes some improvements of Semantic Similarity based Crossover. Firstly, a new way to sample semantics, based on the fitness cases of the problem, is introduced. Then, a number of attrbutes are added to every node in a GP individuals to store semantic resulting in a new GP system called Attributes Genetic Programming (AGP). These attributes speed up the semantic checking in SSC. Next, two methods to overcome the limitation of tuning semantic parameters in SSC are proposed. These methods further improve GP performance.

Chapter 7 examines a number of basic properties of GP under semantic based crossovers. These properties consist of the rate of semantically equivelent subtrees exchanged in crossovers, semantic diversity, semantic locality, the constructive effect of semantic based crossovers, GP code bloat effect and the semantic distance of subtrees exchanged in these crossovers. These analyses shed some light on the improvement performance of semantic based crossovers.

Chapter 8 studies the generalisation ability of semantic based crossovers. The ability of semantic based crossovers to generalise is compared with standard crossover, a validation set based method and bloat control methods. The results show the superior performance

of semantic based crossovers over other methods is not only on training data but also on unseen data.

Chapter 9 aims to answer the question whether semantic locality or syntactic locality of crossover is more important. A new crossover for improving syntactic locality is proposed. The comparison of semantic locality and syntactic locality of crossover is undertaken in three aspects: GP performance, GP code bloat and the ability to generalise. The results show the more useful impact of controlling semantic locality.

The last chapter in Part 2, Chapter 10, is dedicated to investigate the relationship between semantic based operators with problem diffculty and the fitness landscape. Two questions addressed are how semantic based crossovers deal with increasingly difficult problems and how semantic similarity based crossovers affect fitness landscape. The experimental results confirm the superior performance of semantic based crossover when increasing problem difficulty and show their ability to smooth out the fitness landscape of probelms.

In Part 3, Chapter 11, applies the semantic based crossover to Boolean problems. For the Boolean domain, some new crossovers that are inspired from SAC and SSC are proposed. The experiments show the better performance of these new crossovers in comparison with standard crossover and the previous crossovers.

Chapter 12 employs semantic based crossovers to solve time series prediction problems. Two time series are tested. The first one is an artificial one, Mackey time series and the second one is a real-world time series, tide series in Venice Lagoon, Italy. The experimental results show the ability of semantic based crossovers in solving these problems.

In the last chapter – Chapter 13, we sumarise the thesis, review the contributions and present some future directions drawn from this thesis.

Chapter 2

Evolutionary Algorithms and Genetic Programming

This chapter gives an introduction to Evolutionary Algorithms and Genetic Programming. It starts with a brief introduction to Evolutionary Algorithms (EAs) followed by a more detailed introduction to Genetic Programming (GP).

2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of search techniques that are inspired by biological evolution. The resemblance can be seen in the following aspects. EAs solve problems by presenting the solutions through a population of individuals. These individuals compete with each other based on Darwin's principle of natural selection to survive and the evolutionary process is implemented using a number of genetic operators similar to genetic operators in biological genetics. The basic flowchart of an EA is presented in Figure 2.1.

An EA starts with an initial population of candidate solutions. This population is often generated at random, perhaps, with some constraints (depending on the problem requirement and/or the representation of solution). Then, each individual in the population is evaluated and assigned a fitness value. This fitness value represents the ability of the

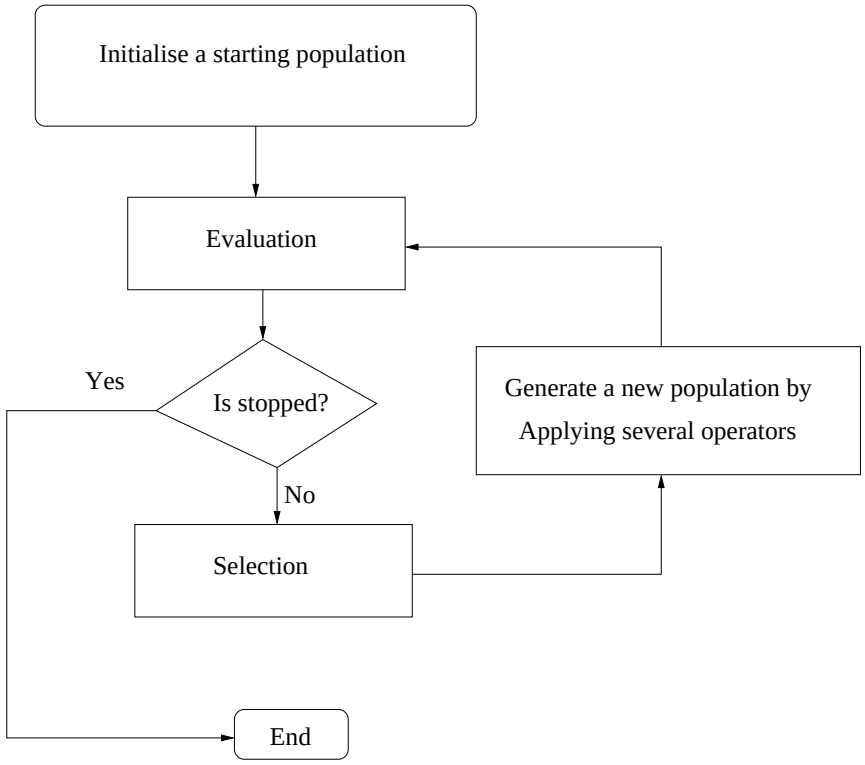

Fig. 2.1: A basic flowchart of Evolutionary Algorithms.

individual to solve the problem. Stopping criteria is then checked to determine if the evolutionary process should be stopped. If they are not satisfied, a number of individuals are selected and added to a mating pool. Next, genetic operators are used to modify the selected individual to generate a new population. This evolutionary process is repeated until the stopping criteria is satisfied.

Since pioneering work of Friedberg [50, 51], Fogel [47], Rechenberg [162], and Holland [71], EAs have largely been developed. The development of EAs can be roughly classified into four main streams.

Evolutionary Programming (EP) was first introduced by Fogel in the 1960s [47]. In this work, he used finite state machines (FSM) as predictors and evolved them. Fixed-length strings are used to represent FSMs and mutation was designed as the main genetic operator.

Evolutionary Strategies (ES) is an optimisation technique based on ideas of adapta-

tion and evolution proposed by Schwefel in the 1960s and 1970s [162]. ES uses a fixed vector of numerical values as its representation of solutions. Mutation of solution is implemented by adding Gaussian noise to the solution. Self-adaption mechanisms are also used to change the variance of the mutation operator.

Genetic Algorithms (GA) is perhaps the most popular form of EAs. It was first introduced by Holland in the 1970s [71]. In an GA, a population in the form of strings of numbers, traditionally binary, is initially (randomly) created then evolved. Two main operators used in GA are crossover and mutation.

Genetic Programming (GP) was popularized by Koza in the 1990s [98]. In GP, solutions of problems are represented in the form of variable shape and size parse trees. Both the structure and the contents of the solution are evolved. This allows computer programs to be encoded and evolved.

In the remainder of this chapter, we only focus on Genetic Programming as it is the algorithm which is the primary focus of this thesis.

2.2 Genetic Programming

Genetic Programming is an evolutionary paradigm that is inspired by biological evolution to find the solutions, in the form of computer programs, for a problem. It can also be seen as a machine learning method to optimize a population of computer programs to perform a given computational task. GP has been recognised by a number of researchers and practitioners as the successor of GA as it inherits a number of important characteristics from GA. While the distinction between GP and GA is philosophical. It is generally agreed that their objective is different: While GA is mostly used for the task of optimisation, GP systems are often used for learning. In GAs, the definition and function is given, the GA systems are used to optimise the parameters of that function. Conversely, GP might be used to learn the definition of the function itself from sampled data. In other words, GA

is usually used to find the optimal parameters of the solution for a problem when the structure of the solution is fixed and known in advance, while GP is preferred if the task is to seek both content and structure of solutions.

The early dawn of computer program evolution can at least be traced back to the 50s of the last century in that Friedberg used computers to learn programs for itself [50, 51]. Later Fogel et al. applied evolutionary algorithms to the problem of discovering finite-state machines, which are in turn simple forms of computer programs [47]. The idea of using chromosome (solution) representation with variable size was probably first proposed by Smith in [176], where the author used individuals of variant size to evolve classifier systems. Cramer [33] introduced more features that are similar to the GP we know today, in which procedural languages are represented in tree-based structures and operated on by suitably defined GA-operators. This work was then greatly expanded by John R. Koza, who has pioneered the application of GP in various complex optimization, learning, and search problems [98, 99, 101].

In the 1990s, GP was mainly tested on relatively simple problems. The reason was that GP was rather computationally intensive. However, due to the recent improvements in GP technology and the exponential growth in Central Processing Unit (CPU) power, GP has been applied to solve many real-world problems. The application of GP includes quantum computing, electronic design, game playing, sorting, to name but a few. In [8, 154, 100], the authors presented a vast range of real-world problems that has been solved by using GP.

Since GP can be seen as an evolutionary algorithm, it shares a number of common characteristics with other EAs. In order to apply GP to solve a problem, The following steps need to be processed [14]:

1. Select a representation, a set of functions and terminals, and a fitness function for the problem.

2. Initialise a population of individuals.

3. Evaluate the fitness of the individuals in the population.

4. If the termination conditions have been reached, exit. Otherwise, go to step 5.

5. Choose a number of individuals (candidate solutions) using a certain selection method.

6. Apply a number of genetic operators on the selected solutions to generate a new population.

7. Repeat from step 3 to step 6.

To start using GP in solving a problem, GP practitioners need to select an appropriate representation format. While tree-based representation is the most popular form, other representations such as linear representation [46, 144], grammar-based representation [191, 122], and graph-based representation [124] can also be used. After that, a set of functions F and terminals T are chosen. The function set $F = \{f_1, f_2, ..., f_n\}$ includes a number 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.

The first step in running a GP system is to create an initial population of candidate solutions. This population is usually randomly generated with respect to some constraints in terms of syntax (max depth of trees or max size of chromosomes). This population is served as the starting point of a GP algorithm. The fitness function is then called to measure the fitness value for each individual in the population. The GP process is finished when the termination condition in step 4 is true. The termination condition is assigned a true value when either the perfect (full score) solution is found or the algorithm exceeds a number of fitness evaluations.

Based on a fitness measure, the fitter solutions (the better solutions to the problem) are selected using a selection method. Next, a new population is generated by applying a number of genetic operators to the chosen individuals. The main operators include crossover, mutation and reproduction. The reproduction operation simply copies a selected individual to the next generation. The mutation operator adds new genetic material to the population by modifying the individual while the crossover operation generates two new

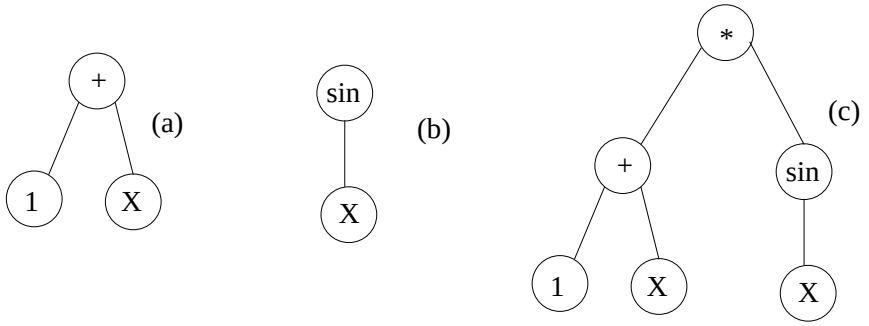

Fig. 2.2: A set of individuals that are constructed from $F = (+, -, *, /, sin, cos, log, exp)$ and $T = (X, 1)$.

individuals by combining two old individuals. These operations will be further described in the following sections.

2.2.1 Representation of Candidate Solutions

Although, there are a number of ways to represent candidate solutions in a GP system, a tree-based representation is still the most popular form [14]. For this reason, this thesis mainly focuses on tree-based representation. Tree-based representation for an individual of an evolutionary algorithm was first proposed by Cramer [33] and then intensively advocated by Koza [98]. To generate a population of individuals, a GP practitioner must first select a function set $F = \{f_1, f_2, ..., f_n\}$ and a terminal set $T = \{t_1, t_2, ..., t_m\}$. The function set might include arithmetic operations (+, - , *, /), mathematical functions (such as sin, cos, log, exp), boolean operations (AND, OR, XOR, NOT), conditional operations (such as IF-THEN-ELSE) and other functions that can be defined for a specific problem. Each function in the function set has a fixed number of arguments, that is considered as its arity. For example, function + has 2-arity, while function *sin* has 1-arity. The terminal set often consists of a number variables and several constants. All terminals have an arity of zero.

When the function and the terminal sets have been decided, a GP individual is built up recursively from these sets. Figure 2.2 shows some individuals that are generated from the function set of $F = (+, -, *, /, sin, cos, log, exp)$ and the terminal set of $T = (X, 1)$.

It is essential that selected function and terminal sets satisfy the *closure* and *sufficiency*

properties [8]. The closure property requires that each function in the function set must gracefully handle every possible input it might receive. In other words, every function must be well defined for any combination of arguments that it may encounter. The reason for this property is that every candidate solution must successfully be executed to have a fitness value. An example of valid function and terminal sets are the function set of $F = \{+, -\}$ and the terminal set of $T = \{X, 0\}$ and an example of invalid sets include the function of $F = \{*, /\}$ and the terminal set of $T = \{X, 0\}$.

The *sufficiency* property requires that the set of functions and the set of terminals must have enough expressive power to represent the solution for the problem. For example, the function set $F = (+, -, *, /, sin, cos, log, exp)$, and the terminal set $T = (X, 1)$ satisfy the sufficiency property in learning real-valued functions. It, however, does not satisfy this property if the learning function is Boolean.

2.2.2 Initialising a Starting Population

Initialisation of the population is the first step of the evolutionary process. It is used to generate a population of tree structure programs that will be evolved in the later steps. There are three main approaches for creating a population of a tree-based GP system: the *grow* method, the *full* method and the *ramped half-and-half* method [98].

The grow initialisation of an individual starts by randomly selecting a function f_i in the function set F. This function becomes the root node of the tree. Let n be the arity of the selected function, then n nodes are randomly chosen from the set of $F \cup T$ as the children of the root node. If a terminal is chosen, this branch of the tree is terminated. If a function is selected, the process is recursively applied for that function. The maximal depth of tree is usually used to limit the size of the initial individual. The grow initialisation of the population uses the grow intialisation for all individuals in the population.

In full initialisation, instead of selecting nodes from $F \cup T$, when constructing a tree only functions from F are chosen until it reaches the maximal depth. At this depth only primitives from T are chosen.

Ramped half-and-half initialisation is the most popular method that is widely used by GP practitioners nowadays. The motivation for it is to avoid the situation of similar trees which can be generated in the two above approaches. This leads to the increase of the *syntactical diversity* of the GP population [154]. The ramped half-and-half initialisation of the population is in fact the combination of the grow and full initialisation in which a half of the population is created by the grow method and the remaining half is generated by the full method.

2.2.3 Fitness Evaluation

Each individual in the population is assigned a numerical value called fitness. The fitness of an individual presents its ability to solve the problem. This value is calculated based on some well-defined procedures. There are two fitness measures that are popularly used in GP, namely *raw* and *standardised* fitness. They are detailed as follows.

Raw fitness can be seen as the most simple form of fitness that reflects the ability of an individual to solve the problem. For example, if the problem is to evolve a classifier of a number of objects, the raw fitness could be defined as the number of objects that are correctly classified. Raw fitness is often evaluated based on a test set of the problem called fitness cases. The fitness cases are usually presented in the form a set of input-output values of the problem. Assume that the fitness cases of a problem consist of N pairs $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, then the raw fitness of an individual i, $f_R(i)$ can be define as follows:

$$f_R(i) = \sum_{j=1}^{N} (g(x_j) - y_j) \tag{2.1}$$

where $g(x_j)$ is the output of the individual with the input is x_j.

Standardised fitness is calculated based on raw fitness so that the smaller (for minimizing problems) is the better. If the raw fitness already has this property the standardised

fitness is the same as the raw fitness. In many situations, it is often convenient that the full score individual has the stardardised fitness of zero.

2.2.4 Selection Operator

Each individual in the population that has been assigned a fitness value has an opportunity to be selected to participate in the breeding of the new population. Over the years, several selection methods have been developed. In this section the three most popular form of selection operations will be described. These selection methods include tournament, fitness proportionate and ranked selection.

Perhaps, tournament selection is the most popular form of the three selection mechanisms. The initial study of tournament selection can be traced back to the early 1980s [17]. In tournament selection, a number of individuals (*tournament size*) are randomly selected from the population. These individuals are compared with each other and the winner (in terms of better fitness) is selected to go to the mating pool. This process is then repeated N times where N is the population size. The advantage of tournament selection is that it allows the adjustment of the selection pressure by tuning the tournament size. A small tournament size leads to a low selection pressure while a large one results in a high selection pressure. Moreover, this method does not require a comparison of the fitness between all individuals. This may help to save a large amount of processing time and provides an easy way to parallelise the algorithms.

The second common form of selection mechanism is fitness proportionate selection [71, 98]. In fitness proportionate selection, each individual is assigned a probability to be selected for the mating pool. Let N be the population size, and $\{f_1, f_2, ..., f_N\}$ be the set of the fitness of individuals in the population, then the probability p_i of individual i to be selected is calculated by the following equation.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.2}$$

This selection mechanism, although has widely been used in GP and GA, has a drawback: Its behaviour strongly depends on the difference between fitnesses of the individuals [15]. If the difference between the fitness of good and bad individuals is high, then it is likely that only the good ones will be chosen, thus decreasing the diversity of the population. Conversely, if the difference of the fitness between them is too small, then this selection method works mostly similar to random selection since the probabilities to be selected of all individuals are almost the same.

The third common form of selection is ranking selection. Ranking selection was first proposed in [58] to soften the potentially dominating impacts of high fitness individuals in the fitness proportionate selection. In ranking selection, all individuals in the population are sorted based on their fitness. The selection probability is then assigned to each individual based on its order in the population. There are two main methods to index individuals in the rank: linear and exponential ranking. Although, this selection technique helps to reduce the weakness of fitness proportionate selection, it has a drawback: in some cases, especially in exponential ranking, it increases the difference between closed fitness individuals so that the better one can be selected more frequently [193].

2.2.5 Crossover

It is well-known that crossover (sexual recombination) is the primary operator for GP [98]. Crossover creates variation in the population by generating new children that consist of parts taken from each parent. Since, there are various representations of GP, crossover for each representation is different from one representation to another. This section only describes the crossover operator for tree-based GP.

In standard crossover (SC), [98], two parents are selected by using a selection method, then one subtree is randomly selected in each parent. A procedure is called to check if these two subtrees are legal for crossover (syntactic closure properties, depth of resulting children,...). If so, the crossover is executed by simply swapping the two chosen subtrees, and the resultant offspring are added to the next generation. Figure 2.3 shows how SC

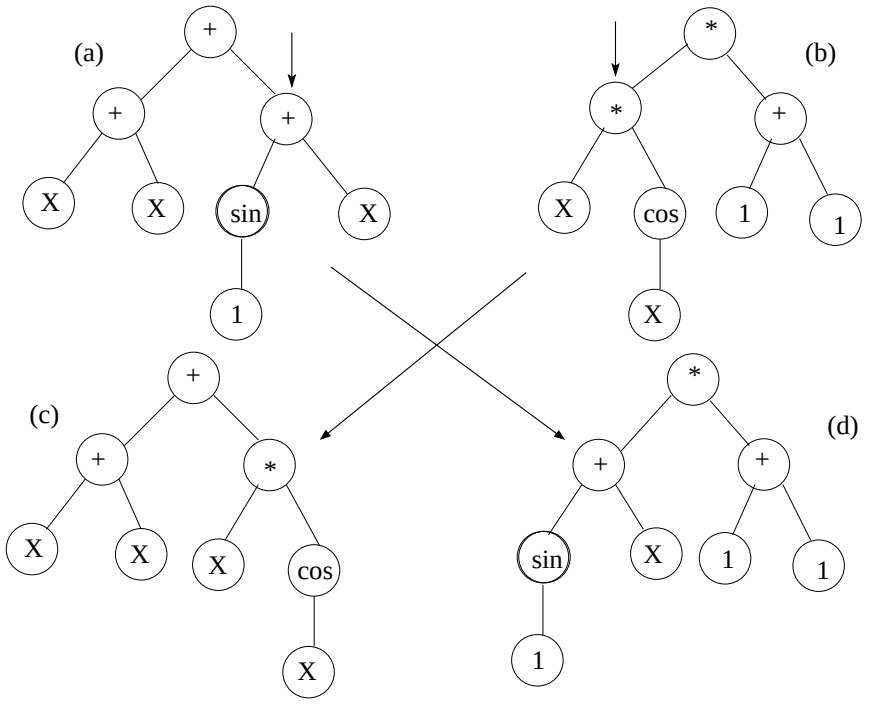

Fig. 2.3: Standard Crossover: parents (top) and their resulting offspring after applying crossover (bottom).

works.

One of the first modifications to the crossover process was proposed by Koza [98], in which the nodes chosen for crossing over is 90% biased to function (internal node) and 10% biased to leaves (terminal nodes). Although this method encourages the exchange of more genetic material (bigger subtrees) between the two participating individuals, it risks exacerbating bloat and thus making it more difficult to refine solutions in later generations [11]. Other improvements of crossover include height-fair crossover of Oppacher [150], one-point and uniform crossover of Poli and Langdon [156, 108], to name but a few. A more detailed review of alternative crossovers in GP will be given in Chapter 3.

2.2.6 Mutation

Mutation can be seen as the secondary operator in GP. While crossover is a sexual recombination, mutation is an asexual operator, meaning that it operates on only one parent. In

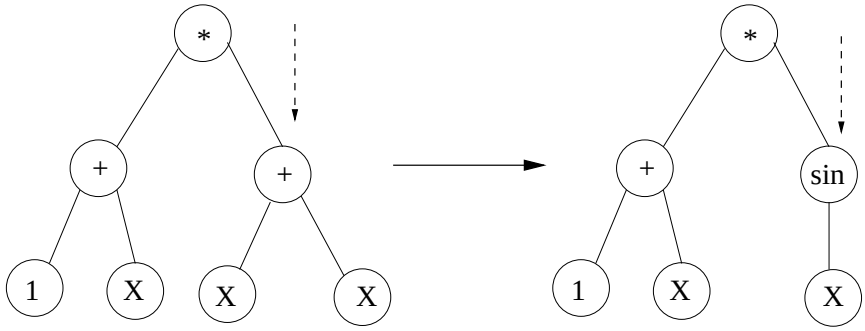

Fig. 2.4: Standard Mutation: parent (left) and its resulting offspring after applying mutation (right).

Standard Mutation (SM) [98], often called subtree mutation, a point is randomly chosen, with uniform probability, within the selected individual. This point is called mutation point. Then the subtree rooted at the mutation point is deleted and a newly generated subtree is added at that point. Figure 2.4 presents how SM works.

Although, Koza introduced mutation as the secondary operator after crossover, the comparative importance of GP crossover and mutation operators is still the subject of much debate. An early argument of this debate was given by Luke and Spector [112] when the authors did an exclusive comparison between crossover and mutation and found that there was very little different performance between these two operators. In a more recent work, White and Poulding [192] conducted an experiment to compare the crossover and mutation in GP with the optimal conditions of the experimental settings for each operator. Their results showed that on only 2 out of the 6 problems examined that GP with crossover was better than GP with mutation. These findings suggest that the mutation operator is also important in GP and any improvement of the mutation operator could potentially lead to the improvement of GP performance. A more detailed review of mutation in GP will be presented in Chapter 3.

2.2.7 GP parameters

Before running a GP system to solve a problem, GP designers need to setup a number of parameters that characterise the evolution. The list of main parameters is the following

one:

- Population size.
- The maximal number of generations.
- The maximal depth of tree in the initial population.
- The maximal depth of tree for the whole evolutionary process.
- Method used to generate the initial population.
- Selection algorithm.
- Crossover type and its probability.
- Mutation type and its probability.
- The maximal depth of tree in the mutation.
- The probability of reproduction.

From the early day, a number of improvements have been introduced to GP. These improvements include elitism selection [37, 158], steady state selection [189] and the use of Automatic Define Functions (ADFs) [99, 45]. Consequently, a number of other parameters are added to a GP system. These parameters comprise the presence or absence of steady state, the number of ADFs and number of each ADF's parameters, the use or not use of elitism. The setting of these parameter impacts to the ability to learn of the GP system. The decision to choose these parameters often depends on the experiments and based on the experience of GP practitioners.

Recently, GP has broadly extended. These extensions comprise alternative operators, grammatical, developmental, and graph tree-based GP, Multi-objective GP. A more detailed discussion of these extensions can be found in [154]. GP has also been applied to solve various problems, and in many situations GP produced competitive results with human [100]. However, a number of problems are still open with GP researchers [147]. These open issues orient future research in GP.

2.3 Conclusion

This chapter presented an introduction to Evolutionary Algorithms and Genetic Programming. It gave a brief introduction to Evolutionary Algorithms, a class of algorithms that are inspired by biological evolution. A more detailed introduction to Genetic Programming was given after that. The discussion of Genetic Programming includes GP initialisation, fitness evaluation, GP operators, and GP parameters. The next chapter will present a review of the related work to the research in this thesis.

Chapter 3

A Review of Related Work

This chapter presents a survey of the related work to research in the thesis. We first review the work on the use of semantic information in Genetic Programming, which is divided into three main strands: Using grammars, using formal methods, and based on tree-based representation. Next, we review the research on alternative operators in GP. These operators include crossover and mutation. Finally, we briefly survey work on diversity and locality in GP.

3.1 Semantics in Genetic Programming

Since Genetic Programming (GP) was born, it has been seen as a potentially powerful method for automated synthesis of computer programs by evolutionary means. The evolutionary process in GP starts with a randomly generated population of programs (individuals). These individuals are then evaluated to calculate their fitness. The fitness of an individual is usually measured by its ability to solve the problem. Next, these individuals are evolved by applying some genetic operators. The operators are designed so that the resulting children are syntactically valid individuals. However, from the perspective of a programmer, this is unusual. Computer programs are not only constrained by syntax but also by semantics. Therefore, a number of researchers have tried to use semantics to aug-

ment GP in solving problems. From our perspective, these attempts can be classified into three categories: using grammars, using formal methods, and using semantics based on GP representation. These methods will be discussed in detail in the following subsections.

3.1.1 Grammars in Genetic Programming

Perhaps, Whigham is one of the first authors who used grammars in GP [151, 152, 153]. The resulting system is called Grammar Guided Genetic Programming (GGGP). In his system, each individual is represented as a derivation tree of a context-free grammar. The use of context-free grammars results in some modifications to traditional GP. These modifications are in population initialization and genetic operators. While the initialization in the traditional GP is done almost randomly, in GGGP it is done in a more strict manner so that any generated individual is always a derivation tree of the defining grammar. In addition, crossover in GGGP can only swap two subtrees, of which the roots are labeled with the same non-terminal symbol. More details about the differences between GP and GGGP can be found in [152].

Using grammars in GP produces many benefits. First, it helps to ensure the property of closure [152]. Also, the use of grammars helps to encode domain knowledge on the syntactical structure of programs. Additionally, by using grammars, it is convenient to re-bias the syntactical structure of programs by changing grammars. More details about these benefits of using grammars in GP can be found in [67].

Despite a number of benefits, using context-free grammars does not help to incorporate semantic information into GP. Thus, some researchers have attempted to go beyond context-free formalisms to achieve this goal. These grammars include attribute grammars, logic grammars and some others. They are briefly discussed as follows.

Attribute Grammars in GP

Attribute grammars were first defined by Donald Knuth in 1968 as means for formalizing semantics of context-free languages [97]. An attribute grammar may be considered as an extension of a context-free grammar using a set of attributes to provide context sensitivity.

<expr>::=<expr> + <expr>
| <expr> − <expr>
| <expr> ∗ <expr>
|(<expr>)
| <pre_op> (<expr>)
| <var>
<pre op>::=sin
|cos
|exp
|log
<var>::=x

Fig. 3.1: The context-free grammar.

Each distinct symbol in an attribute grammar may have a finite, possibly empty a set of attributes. Each attribute has a domain of values. The evaluation rules and the conditions may be associated with each symbol to change value of its attributes or to check if the derivation tree is a valid tree or not [97].

Perhaps, using attribute grammars as a way to add semantics into GP only started recently, although Talib S. Hussain and Roger A. Browse have used an attribute grammar to represent a neural network to evolve a neural network [74, 75, 76]. A study which shows a comprehensive comparison between a GP system with and without semantics information by using attribute grammars was done by Echeanda and his colleagues [35]. In this paper, the authors compared the performance of a GP system, namely Grammatical Evolution (GE) [144, 145], when semantic information is used and not used. By using an attribute grammar, they could check if individuals generated by GE's genotype-phenotype mapping are valid both in terms of syntax and semantics. The work showed that using semantic information could improve the performance of GE on a symbolic regression problem. The target function chosen in [35] was $f(x) = x^4 + x^3 + x^2 + x$. The context-free and attribute grammars used in their work are shown in Figure 3.1 and Figure 3.2.

The attribute grammar in Figure 3.2 is very similar to the context-free grammar in Figure 3.1. The main difference is the existence of 21 attributes (v_i, i=0,...,20) to record the value of the expression on each control point. The semantic information is used in

<expr>::= <expr>$_1$+<expr>$_2$	$< expr > .v_i =< expr >_1 .v_i + < expr >_2 .v_i$
| <expr>$_1$-<expr>$_2$	$< expr > .v_i =< expr >_1 .v_i - < expr >_2 .v_i$
| <expr>$_1$*<expr>$_2$	$< expr > .v_i =< expr >_1 .v_i * < expr >_2 .v_i$
|(<expr>$_1$)	$< expr > .v_i =< expr >_1 .v_i$
| <pre_op>(<expr>$_1$)	$< expr > .v_i =< pre\_op > .f(< expr >_1 .v_i)$
| <var>	$< expr > .v_i =< var > .v_i$
<pre_op>::=sin	$< pre\_op > .f = sin$
|cos	$< pre\_op > .f = cos$
|exp	$< pre\_op > .f = exp$
|log	$< pre\_op > .f = log$
<var>::=x	$< var > .v_0 = -1$
	$< var > .v_1 = -0.9$
	$< var > .v_2 = -0.8$
	...
	$< var > .v_{20} = 1$

Fig. 3.2: The attribute grammar.

this problem to check if an individual exactly fits the target function on some sample points (three points, -1, 0, 1 are checked in the experiment in the paper). If an individual does not fit the target function on these points, it will be removed from the population or has a very bad fitness value. By adding this semantic information, the authors showed that it can improve the performance of GE. The paper also pointed out that adding too much semantic information (checking in too many points of sample points), can cause bad performance. This way of using semantics is straight forward and easy to implement although this information is quite similar to fitness information.

At the same time as [35], Robert Cleary and his collaborators proposed the use of attribute grammars to help GE in solving 0/1 multi-constrained knapsack problem [146, 26, 25]. In this work, they used attribute grammars in two ways. Firstly, attribute grammars were used to overcome the limitation of a context-free grammar in decoding solutions for this problem. By adding attributes to a context-free grammar to provide the system with context sensitive capacity, they could avoid the duplication of items in the phenotype for this problem. Secondly, their attribute grammar was also used to check semantic information during the process of the genotype–phenotype mapping. In this case, an attribute called *overweight* was used to govern the total weight in the knapsack. The

experiment results showed that the GE coupled with an attribute grammar, outperforms the traditional version (using context free grammars) and some other methods such as Genetic Algorithm and Hybrid Genetic Algorithm [93, 30] on the problem.

An extension of attribute grammars is Christiansen grammars. In fact, a Christiansen grammar is an attribute grammar where the first attribute associated to every symbol is also a Christiansen grammar. Christiansen grammars are adaptable grammars in that its rule set can change during the derivation process. More detail about Christiansen grammars can be found in [24, 173].

Rosal et al. [166] used a Christiansen grammar to add semantics to GE (the system was called Christiansen Grammatical Evolution - CGE) to solve some Boolean problems. The semantic information stored in the attributes of the grammar is used to change the rules of the grammar during the derivation process. The results from the experiments showed that CGE outperforms traditional grammatical evolution especially on some problems which are difficult with GE using a context-free grammar.

Logic Grammars in GP

Apart from attribute grammars, logic grammars have also been used as a way to incorporate semantics into GP. A logic grammar, also called a definite clause grammar, is a context-free grammar decorated with context sensitive capacity. Logic grammars are rather similar to attribute grammars except in their notation. In logic grammars, each symbol, including both terminal and non-terminal may have arguments [128]. The arguments can be a logical variable, a function or a constant. These variables, functions and constants are called *terms*. A variable has the form of a question mark ? followed by a string of letters and/or digits. For example ?x is a variable of the grammar in Figure 3.3. A function is represented by a symbol followed by a bracketed n-tuple of terms. A constant is simply a 0-arity function. Arguments can be used in a logic grammar to enforce context-dependency or to construct representation "meaning" in the parsing tree. The grammar in Figure 3.3 shows an example of a logic grammar. This grammar is a reduced version of the grammar in [194], where we remove some unnecessary rules for the purpose of simplification.

As the decorated arguments in logic grammers play a similar role to attributes in at-

```
1: start::= member(?x,[X, Y]), [(*], exp-1(?x), exp-1(?x), [)].
2: start::=member(?x,[X, Y]), [(/], exp-1(?x), exp-1(?x), [)].
3: exp-1(?x)::=random(0,1,?y), [(+ ?x ?y)].
4: exp-1(?x)::=random(0,1,?y), [(- ?x ?y)].
5: exp-1(?x)::= [(+ (- Y 11) 12)].
```

Fig. 3.3: A logic grammar.

tribute grammars, they can be used to add semantic information into genetic programming. Man Leung Wong et al. [197, 196, 194, 195, 198] proposed a new version of GP by combining logic grammars with GP, the system is called LOGENPRO (The Logic Grammars Based Genetic Programming System). In this system, a logic grammar is used to determine if individuals generated during initialization or through the application of genetic operators are valid in terms of semantics.

The LOGENPRO system was then applied to a number of problems including learning functional programs [194], learning logic programs [195], learning logic programs from imperfect data [196]. In each problem, the semantic information was used to check if an individual created from the initialisation step or from executing genetic operators is valid under the conditions of the respective grammar. The result from the experiments showed that by using logic grammars to check semantic information, the performance of LOGENPRO is superior than traditional GP.

Overall, there is a common theme in using grammars, regardless of attribute grammars or logic grammars, as a way to incorporate semantic information into genetic programming is that attributes are employed to encode semantics into the formalisms. However, what are the kinds of attributes and how they are encoded into the grammars depend on the type of semantic information that are useful for the target problem.

3.1.2 Formal Methods in Genetic Programming

The use grammars in GP has been extensively studied, while the use formal methods as a way to incorporate semantics in GP has only been raised recently. Formal methods are a class of mathematically based techniques for the specification, development and verification

of software and hardware systems [66]. By using formal methods, ones can gather internal information (in mathematical forms) about the systems. This information is useful for the system design and verification.

Perhaps the first researcher who pioneered this area of research for GP is Colin Johnson. In a series of work, Johnson has advocated for the use of formal methods in the evolutionary process of GP [82, 83, 84, 85]. In [84], Johnson elaborated some directions in which formal methods can be used to improve the performance of GP. His argument was that almost every formal technique can be used to enhance the ability of GP in solving problems. These techniques include: static analysis, model checking, program transformation, and partial evaluation, to name but a few. However, there are only two kinds of techniques that have been used in GP, abstract interpretation and model checking.

Abstract Interpretation with GP

Abstract interpretation is a class of techniques for gathering approximate semantic information of a program [129, 141, 31, 32]. It is one of the main streams of static analysis [16] differing from others, such as data-flow analysis, in that abstract interpretation is conducted not on concrete data of a system, but on an abstract version of this data. In abstract interpretation, the data processed by a program and various operators are abstracted into a set of properties of interest. The analysis of a program consists of checking if these properties are held at some particular points of the program or through the whole program.

Using abstract interpretation, we can deduce information about some interesting properties of a program. This information can be used in different ways in GP. One way is to use this information as a measure of the fitness as in [82, 83]. In this work, Johnson used an interval analysis technique to solve some problems. An interval analysis technique is a kind of abstract interpretation, which is used to infer the extreme values of the variables of a program. A short example, if x belongs a interval [a,b], y belongs a interval [c,d], then the value of z after excusing the statement z=x+y will belong the interval [a+c, b+d]. More details about interval analysis can be found in [161].

Johnson [82] used an interval analysis technique to induce the programs to solve a

placement problem. The aim was to find a placement of a number of rectangles on a region so that they satisfy some desired relations between them. An example relation is Rectangle A must be always on the left of Rectangle B. With this kind of problem, it is very difficult to use traditional fitness measures that are based on a set of sample cases. Firstly, generating a set of sample cases is not easy in this situation. Secondly, even if a set of sample cases can be created, we can not guarantee that the desired constraints will always be satisfied as the list of sample cases can not cover all situation. Thus, the fitness function in [82] is based on interval analysis.

The system starts from a randomly generated population of programs [82]. These programs are evolved through a number of generations. At each generation, an interval analysis technique is used to track the extreme values of the variables. These extreme values are then compared with the required constraint values to see how many desired constraints are satisfied. If a program satisfies more conditions it is assigned a better fitness value, therefore, it has a greater chance of entering to the next generation.

Johnson [83] used an interval analysis technique to solve a problem of controlling the movement of a robot. The objective was to evolve a computer program to control the movement of a robot so that it is as close as possible to another robot (called the target robot) and it is always on the right of a barrier. In this experiment, the fitness function includes two components: one is based on test cases (as close as possible to the target robot) and another is based on a constraint condition (always on the right of the barrier). At each generation during the evolutionary process, the program is analysed, by using an interval analysis technique, to determine if it is satisfied by the constraint condition (the robot is always on the right of the barrier). If it is, the fitness of that program based on training data is multiplied three. In this way, the satisfied programs have a greater chance of being selected.

The work of Johnson demonstrated how interval analysis can be used as a way of measuring fitness [82] or in combining with a traditional fitness measure [83]. However, the experiments in these work are relatively simple and no comparison of these systems with other GP systems has been provided.

While Jonhson used interval analysis as a way to measure the fitness of individuals in GP, Keijzer has used interval analysis in a different way. Keijzer [92] used interval analysis to check if an individual can be undefined in the whole range of input values. For example, if an individual contains function log(x) and by using interval analysis, one can infer that the interval of variable x can contain negative values, this individual will be considered as an undefined individual. If this case happens, the individual will be assigned the worst fitness or simply be deleted from the population. By doing that, Keijzer argued that GP can avoid the disruption of using protected operators. This technique was then used in combination with linear scaling (a technique that is used for smoothing out the fitness function on data training [78]). The system was applied to a class of symbolic regression problems and the experiment results showed that the system could find the solutions that are not only good on training but also on test data (i.e better generalization capacity) [92].

Model Checking in GP

Model checking [39, 7] is a class of techniques for checking if some properties of a system are held. The properties, which need to be checked are presented in the form of temporal logic formulas [44]. These formulas state how variables and states in the systems change with time. There are three steps in performing model checking: modeling, specification, and verification. Modeling is transferring the system into a formalism. The formal model often used is a transition system (TS). Specification involves expressing the properties of interest into temporal logic formulas [44]. Verification is done by using an algorithm to check if a given transition system satisfy a temporal logic formula.

Model checking has also been used as a way of measuring the fitness of systems in GP. Johnson [85] used model checking in cooperation with GP to evolve coffee vending machines. The machine is specified by a number of computation tree logic formulas. The GP system is started by randomly generating a number of machines (individuals). A model checking is then used to check if these machines satisfy the above temporal logic formulas. The fitness function is measured by counting the number of satisfied formulas. If an individual satisfies more propositions, it has a greater fitness value and hence has a greater probability to be selected. Obviously, in this problem, it is very difficult to

apply traditional fitness measures. Therefore, using model checking to measure fitness is reasonable. However, the drawback of this work is that the fitness function does not to be smooth. The reason is that a formula, which is nearly satisfied, will be considered as an absolutely unsatisfied formula. This weakness will be considered by some later research.

Following Johnson's work, Katz and his colleagues [89, 90] used GP with model checking to generate algorithms for a mutual exclusion problem [106]. The work of Katz et al. is different from that of Johnson in two ways. First, Katz and his colleagues used a linear temporal logic for the system specification instead of using a computation tree logic. Hence, the model checking algorithm, which is used for verifying the system, is also different. In these works, the algorithm checked if a path in the graph that represents the behavior of the checked system, is satisfied by the formula. Second, the fitness function did not simply count on how many formulas are satisfied but it was based on a lower level. The scores that are given to a checked formula was divided into 4 levels. Level 0 was allocated when there is no satisfied path in the graph. Level 1 was assigned when there are some satisfied paths and the program can reach a state from which the formula is not satisfied. Level 3 was when all paths are satisfied and the program can only reach the unsatisfied state when there is an infinite hostile scheduler (which is used in these works). Level 4 was assigned when the formula is always satisfied. The fitness value was calculated by summing up value of all assigned scores in the above step. Besides, they also prioritize the properties. Some more important properties are checked first. If these properties are not satisfied, then all properties left will be assigned to level 0. By making a smoother fitness function, the authors claimed that it could provide a higher probability of convergence.

We conclude this section by highlighting some advantages and disadvantages of using formal methods in GP. The advantage of formal methods lies in their rigorous mathematical foundations, potentially helping GP to evolve computer programs. However they are high in complexity and difficult to implement, possibly explaining the limited number of related publications since the advocacy of Johnson [83]. Their main applications to date has mainly been in evolving control strategies.

3.1.3 Semantic Based on Tree Representation

Apart from the approaches based on grammars and formal methods, semantics can also be directly calculated/extracted on/from the tree-based representation of GP. The way in which semantics can be extracted from tree-based representation depends on the problem. In Boolean problems, semantics can be accurately calculated in different ways. Beadle and Johnson [11] calculated the semantic information on Boolean trees by firstly reducing the trees using ordered binary decision diagrams (ROBDDs) [18]. A ordered binary decision diagram is a binary decision diagram in which the order of the label between the nodes is respected. It means that the label of a node is always greater than the label of its children. A binary decision diagram is a directed acyclic tree, also called a *dag*, which is obtained by applying some reduction rules on a binary decision tree. Two reduction rules are [174]:

- Elimination of redundant nodes: if there exists a node n such that the left subtree and the right subtree of n are the same, then remove this note and replace it by their left or right subtree.
- Merging isomorphic subtrees. If two subtrees rooted at two different nodes n_1, n_2 are isomorphic, then merge them into one, i.e., remove the subtree rooted at n_2 and redirect all angles to n_2 by angles to n_1.

Figure 3.4(e) is an example of a binary decision diagram that is obtained by applying the two above reduction rules on the binary decision tree on Figure 3.4(a). In this example, Figure (b) is obtained from (a) by merging the subtrees rooted at r. Figure (c) is obtained by removing a redundant node on r from (b). Figure (d) is obtained from (c) by merging isomorphic subtrees rooted at p. Finally, Figure (e) is obtained by removing a redundant node rooted at p from (d).

After reducing from Boolean trees using ROBDDs, the equivalence in term of semantics of the two trees are compared [11]. This semantic equivalence checking is then used to determine which generated individuals are copied to the next generation. If the offspring are semantically equivalent to their parents, they are discarded and the crossover is restarted.

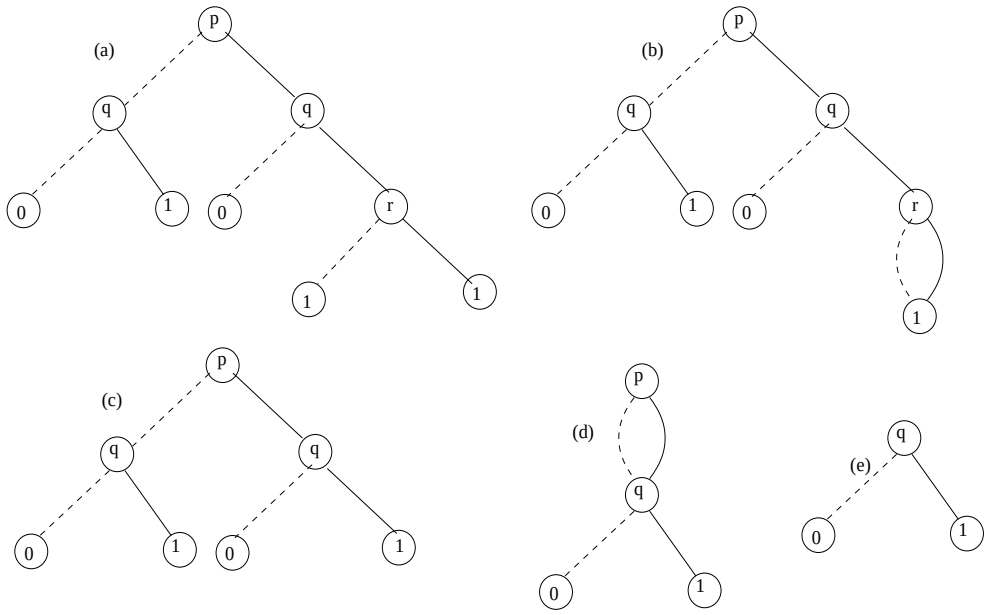

Fig. 3.4: Transformation of a binary decision tree into a BDD.

This process is repeated until semantically new children are found. The authors argued that this results in increased semantic diversity in the evolving population, and a consequent improvement in GP performance. This method of semantic equivalence checking is also applied to drive mutation [12] and to guide the initialisation phase of GP [13], where the authors show that it benefits for GP in both phases.

While Beadle and Johnson compared semantics of two Boolean tree expressions through ROBDDs, McPhee et al. extracted semantic information from Boolean expression trees by enumerating all possible inputs of each individual [123]. In this method, they considered semantics of two components in each tree: subtrees and contexts. A context is the remainder of a tree after removing a subtree at one point in that tree. While the semantics of a subtree can easily be computed by enumerating all possible input values, the semantics of a context as in Figure 3.5 depends on three components:

- The operator g immediately above the insertion point.
- The semantics of the context obtained by removing the subtree rooted at g (the Parent semantics).

- The subtree semantics of the other argument(x) of the operator g.

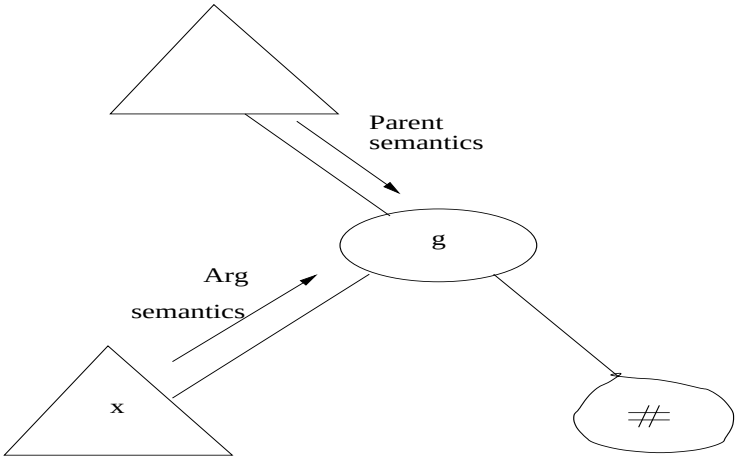

Fig. 3.5: A context which is obtained by removing a subtree rooted at node c.

By considering both of these semantics, McPhee at el. conducted experiments to test the change of these semantic components during the evolutionary process of GP. The result from their experiment showed that there are a large number of fixed semantic subtrees when the size of trees is increased during the search process. Here, a subtree is called a fixed semantic subtree if the semantics of the tree does not change when this subtree is replaced by any other subtrees. They hypothesized that it is very difficult to change the semantics of trees when the size of these trees is increased. Therefore, genetic operators including crossover and mutation have a very small effect in this situation [123].

A similar method was also used as a way of measuring fitness of an individual [102]. In this work, by considering the semantics of context, one can know how the children will be if the crossover is executed on the point where this subtree is removed. This information will be used as a fitness measure. This is called the *potential fitness function* [102]. The method was applied to a class of Boolean problems and the result from the experiment showed that it helps to improve the performance of GP in comparison with the traditional fitness measure.

For real valued function regression problems, calculating the exact semantics is infeasible. Thus, approximate semantics must be used instead. One method for measuring

approximate semantics in this problem is proposed in [127]. In this paper, the equivalent semantics of two individuals (two trees) is decided by comparing their outputs on a number of random sampled points in the domain. Two trees are equivalent if the mean square error of them on a number of random sampled points is less than a small value. This equivalence was then used to simplify the trees during the evolutionary process. The result from the experimented shows that this semantic information is useful for simplifying GP individual for the tested problems [127].

Recently, Krawiec and Lichocki proposed a way to measure the semantics of an individual based on fitness cases [103]. In this work, the semantics of an individual is defined as a vector of which each element is the output of the individual at the corresponding input fitness case. This semantics was used to guide crossover in a method similar to *Soft Brood Selection* SBS [3], known as *Approximating Geometric Crossover* (AGC). In AGC, a number of children are generated by a crossover operation, the children that are most similar to their parents – in terms of semantics – being added to the next generation. The experiments were conducted on both real valued and boolean regression problems and the results showed that AGC is no better than SC on real valued problems, and only slightly superior to SC on Boolean ones [104]. The same kind of semantics was then used to build functional modulation for GP, for which the experimental results showed that it may be useful in characterising the compositionality and difficulty of a problem, potentially leading to performance improvements for GP [104].

3.2 Alternative Operators in Genetic Programming

This section presents an overview of alternative crossovers and mutation in GP.

3.2.1 Alternative Crossovers in Genetic Programming

It is well-known that crossover is the primary operator for GP [98]. In the standard (original) crossover (SC) [98], two parents are selected, and then one subtree is randomly

selected in each parent. A procedure is called to check if these two subtrees are legal for crossover (syntactic closure property, depth of resulting children,...). If so, the crossover is executed by simply swapping the two chosen subtrees, and the resultant offspring are added to the next generation.

Much research has concentrated on the efficiency of crossover, resulting in new and improved operators which might be classified into three categories as follows:

1. crossovers based on syntax (structure)
2. crossovers based on context
3. crossovers based on semantics

Most of the early modifications to SC were based on syntax [98, 156, 108, 150, 79]. Koza [98] proposed a crossover that is 90% biased to function nodes and 10% biased to terminal nodes as crossover points. Although this method encourages the exchange of more genetic material (bigger subtrees) between the two participating individuals, it risks exacerbating bloat and thus making it more difficult to refine solutions in later generations [11]. O'Reilly and Oppacher [150] introduced a height-fair crossover, in which all subtree heights in the two parents are recorded, and one subtree height is randomly selected. The crossover sites in both parents are then restricted to that particular height. Ito et al. [79] presented a similar depth-dependent crossover, aiming to preserve building blocks. In this method, the probability of selecting a node is biased towards the root – nodes that are near the root have a greater probability to be selected for crossover operation. The bias of the selection probability is set by the user, and it is left unchanged during the search process. However it is not robust: if it is not carefully set for a particular problem, the performance can be very poor [80]. Poli and Langdon [156, 108] introduced one-point crossover and uniform crossover. In these methods, when two parents are selected for crossover, they are aligned based on their shapes. By aligning two parents, the common shape of these parents (starting from the roots) can be determined. The crossover points are then randomly selected from the nodes that lie in the common shape region. This kind of crossover has been shown

especially effective on Boolean problems as it causes a bigger genetic material exchange in earlier generations (in these generations the common shape is often very small) and yet can tune the solutions in later generations (when the common shapes are bigger) [156, 108].

More recently, context has been used as extra information for the selection of crossover points [3, 180, 179, 64, 116]. This class of crossover operators is perhaps closest to semantic based crossovers. Altenberg [3] proposed a new crossover inspired by the observation that in most animal species, breeding occurs more often than the number of surviving offsprings might suggest. In other words, viable offspring are not always produced as a consequence of breeding. This crossover is called a *Soft Brood Selection* (SBS). Two parents are selected for crossover, then N random crossover operations are performed to generate a *Brood* of $2N$ children. The children are evaluated and then sorted based on their fitness. The two best children are copied to the next generation, the rest are discarded. This crossover was then developed by Tackett [180, 179] by using a subset of fitness cases to figure out which children in the *Brood* are added to the next generation. Hengpraprohm and Chongstitvatana [64] proposed *Selective Crossover*, in which each subtree is assigned an impact value, reflecting how well (or badly) the subtree affects the containing tree. The impact of a subtree is determined by removing that subtree and replacing it with a random terminal node. The change in fitness of the resultant individual is the impact value. The crossover operator is performed by replacing the worst subtree of one parent with the best subtree of the other. Majeed and Ryan [116] proposed *Context Aware Crossover* (CAC). In CAC, after the two parents have been selected for crossover operation, one subtree is randomly chosen in the first. This subtree is then crossed over into all possible locations in the second, then all generated children are evaluated. The best child (based on fitness) is selected as the result, and copied to the next generation. This process is then repeated for the second parent. The advantage of this context-based crossover is the increased probability of producing better children. On the other hand, it can be very time consuming to evaluate the context of each subtree.

To the best of our knowledge, the only previous use of semantics in crossover are those previously discussed in Section 3.1. They include Beadle and Johnson's [11] Semantics

Driven Crossover for Boolean problems, Krawiec and Lichocki's [103] Approximating Geometric Crossover.

3.2.2 Alternative Mutations in Genetic Programming

Mutation is often seen as the secondary operator in GP. In Standard Mutation (SM) [98] (called *subtree* mutation), an individual is selected by applying one of the selection operator. A random subtree is then chosen in this selected individual (the root of this subtree is called *mutation point*). After that, the chosen subtree is removed from the individual and a new subtree is generated. This new subtree is then adjoined to the mutation point and the individual is added to the next generation.

Although, Koza introduced mutation as the secondary operator after crossover, the relative importance of GP crossover and mutation operators is still the subject of much dispute [112, 192]. A number of studies compare the roles of crossover and mutation in GP [112, 6, 113, 192]. This work suggests that the mutation operator is also important for GP search and any improvement of the mutation operator could potentially lead to the improvement of GP performance. Therefore, a number of variants of mutations have been proposed. Like crossovers, these mutations can also be classified into three categories.

1. Syntax (structure) based mutations.

2. Context based mutations.

3. Semantic based mutations.

Most of the early modifications to standard subtree mutation was purely based on syntax with an aim to reduce code bloat. An example of such modifications is to restrict the standard subtree mutation so that the increase of the program/tree depth after being mutated is no more than 15% of its current depth [94], or must be size-fair [107]. Another mutation to reduce program size is the shrink mutation [5]. In this mutation, a random selected subtree is replaced by a terminal. Angeline [5] used this mutation operator to help

his examination into the sensitivity of the frequency of leaf selection in GP and he showed that it also reduce individual size.

Further modifications of mutation on a syntax level include point mutation and hoist mutation. In point mutation, a node is randomly selected and replaced by another node which has the same number of children (the equivalent arity) [120]. This mutation is inspired by a single bit flit mutation in Genetic Algorithms. This kind of mutation has been used with different success when solving problems [98, 119]. In hoist mutation, a subtree is randomly selected from an individual and this subtree is used to replace the individual from which the subtree is copied [96]. However, this kind of mutation can be highly destructive because of the loss of root functionality, so it could decrease the performance of GP.

In terms of using context as extra information for doing mutation, Majeed and Ryan [117] proposed Context Aware Mutation. This mutation is inspired from their previous work on Context Aware Crossover [116]. In this mutation, a repository of potentially useful subtrees is generated and stored. The mutation is done by randomly choosing a subtree from the repository. This subtree is then used to replace for all possible subtrees in an selected individuals. All generated children are fitness evaluated and the best child (based on fitness) is copied to the next generation. The advantage of this mutation is that it increases the constructive rate of mutation. However, it is very time consuming to evaluate the context of all subtrees in an individual.

Using semantics to guide mutation has only recently been paid attention to. Beadle and Johnson [12] proposed Semantically Driven Mutation (SDM), which is inspired from their previous work in Semantically Driven Crossover [11]. SDM works by checking if the semantics of the generated child is equivalent with its parent. If a repetition happens, the mutation is repeated by randomly generating a new subtree and selecting a new mutation point. SDM has been applied to Boolean and Articial Ant problems and the experimental results showed it helps to improve GP performance.

3.3 Diversity in Genetic Programming

It is widely admitted that maintaining high population diversity is very important in the field of GP [60]. A higher diversity maintaining system often finds new solutions more easily than one with lower diversity. The quick loss of diversity was suggested to mark the premature beginning of a local search phase in GP [53, 157]. GP systems could be trapped into local optima because of premature loss of diversity. When considering the diversity in GP populations, two types of diversity should be distinguished [13]. The first type is syntactic or genotypic diversity, that is, programs in the population being different in terms of syntax. The second type is behavioural or phenotypic diversity, that is, diversity of the behaviour with respect to a set of input-output values. In this thesis, we will regard the second type as semantic diversity. We argues that the second type of diversity is at a more general level than the first one as we will see in Chapter 4 that two programs that are syntactically different, yet they can have identical semantics.

Controlling diversity syntactically has been considered since the early days of GP. Much earlier work focused on the initialisation phase of GP. Koza [98] introduced the well-known Ramped-Half-and-Half technique for creating the initial GP population to reduce the occurrence of duplicated trees. O'Reilly and Oppacher [150] and Poli and Langdon [157] tested various crossover operators to study their impact on syntactic diversity. They showed that standard crossover often leads to loss of diversity, hence is not an ideal operator.

Quite early, Rosca [167] proposed measuring semantic diversity using phenotype entropy. Langdon [107] used (explicit) fitness sharing to preserve diversity. It clusters the population into a number of groups, based on their similarity with respect to a fitness-based metric. Members of the same group are penalized by having to share fitness, while isolated individuals retain the full reward. McKay [121] used implicit fitness sharing, in which the reward for each fitness case is shared by all individuals that give the same output.

Recently, semantic diversity has received more attention from GP researchers. Burke et al. [20] conducted an analysis of different diversity measures to fitness. The experimental results showed that there is a strong correlation of entropy and edit distance with the change

of fitness. Gustafson et al. [60] examined the ability of sampling both unique structures and behaviours in GP. The behaviour sampling results helped to explain previous diversity research and suggested new ways to improve search. Similarly, Looks [111] proposed a new method for sampling semantic-unique individuals in GP. The idea is to generate a number of unique minimal programs, and the population is then generated combining several random programs to these minimal programs. The author argued that it helps to increase the behavioral diversity of the population and leads to significant improvements in the GP performance. The idea of sampling semantically unique individuals in the initialisation phase was then enforced by Beadle and Johnson [13]. Here, the authors used the Ramped-Half-and-Half technique coupled with a semantic equivalent checking procedure using ROBDDs. An individual generated by a Ramped-Half-and-Half technique is semantically compared to all generated individuals. If it is equivalent with one of the previously generated individuals, the individual is discarded from the population. This process is repeated until there is enough desired individuals in the population.

With respect to the semantic diversity of crossover operators, Gustafson et al. [62] proposed a new scheme for selecting parents in crossover that is called *No Same Mate* (NSM) selection. The idea behinds NSM selection is to prevent the crossing-over of two individuals with the same fitness. It was inspired from the author's observation that two individuals with identical fitness likely generate two children with unchanged fitness. The experimental results on the symbolic regression problems were positive. Besides, there are only those discussed in the previous section. They include Beadle and Johnson's [11] Semantics Driven Crossover for Boolean problems, and their [12] Semantic Driven Mutation for Boolean and Santa Fe Ant problems. Overall, promoting diversity especially semantic diversity is important and often leads to the benefit results. It should, however, be noted that, in some problems, maintaining high diversity may not be enough to guarantee improvements in GP performance [19].

3.4 Locality in Genetic Programming

In the field of GP in particular and Evolutionary Computation (EC) in general, locality (small change in genotype corresponding to small change in phenotype) plays a crucial role in the efficiency of an algorithm [57, 69, 169, 171]. Intuitively, maintaining diversity should be accompanied with improving locality. However, it should be noted that most of the previous work on locality in GP (such as [69]) only focused on the syntactic aspect of GP genotype-phenotype mapping. Semantic locality has hardly been investigated.

Rothlauf is one of the early advocators for locality in representation for Evolutionary Algorithms (EAs) [171]. To determine the degree of locality of a genoptype-phenotype mapping, two metrics defined on both genotypic and phenotypic spaces are needed. Rothlauf distinguished two types of locality of a representation: low and high locality. A representation is of high locality if a small change in genotype corresponds to a small change in phenotype. Conversely, the representation is of low locality. The author also mentioned that a representation that has high locality is necessary for an efficient evolutionary search [171].

Although, a representation with high locality is strongly desirable for the search ability of the algorithms, often, designing such a representation is not a trivial task. Most of current GP representations are of low locality, meaning that a small syntactic change can cause a big or even uncontrollable change in phenotypes. For instance, Rothlauf and Oetzel [170] showed that a GP system with genotype-to-phenotype mapping as Grammatical Evolution (GE) [144] could have low locality. To the best of our knowledge, there has only been a GP representation given in [69] which was formally proven to be of a high locality but just in terms of syntax. In this thesis, we extend the concept of locality of a representation of Rothlauf in [171] to the locality of a genetic search operator. We consider the locality in terms of semantic locality rather than syntactic locality as in the previous work in the literature. Moreover, instead of using two discrete values for measuring low and high locality [171], semantic locality in this thesis has a continuous domain of values. We also demonstrate that by using a semantic-tree based representation, the design of crossover

operators that achieve higher semantic locality than standard crossover is possible and this leads to an overall improvement in GP performance.

3.5 Conclusion

This chapter has presented a review of related work to the research in this thesis. Firstly, the methods in which semantics has been used in GP is surveyed. These methods can be divided into three categories: Using grammars, using formal methods and extracting semantics based on GP tree based representation. Then, different operators in GP were discussed. These operators (crossovers and mutations) are grouped into three strands: operators based on syntax, operators based on context, and operators based on semantics. Last, the state of art of diversity and locality research in GP is highlighted. The next chapter will present novel methods that promote semantic diversity and semantic locality of operators.

Chapter 4

Methods

This chapter presents the foundation of the thesis, proposing a novel way to measure semantics of a subtree by sampling a number of points from a problem domain. A semantic distance and two semantic relationships are defined based on this notion of semantics. Last, a number of operators to promote semantic diversity and operators to improve semantic locality are presented. All will be detailed in the following sections.

4.1 Measuring Semantics

This section proposes a new way to measure the semantics of a subtree. This semantics is called *Sampling Semantics*. Sampling Semantics of a subtree is estimated based on sampling a number of points from the problem domain. Following this, a semantic distance between two subtrees and two semantic relationships are defined.

4.1.1 Sampling Semantics

Semantics is a broad concept that has been studied in a number of different fields including Natural Language [2], Psychology [27] and Computer Science [142] among others. While the precise meaning varies from field to field, semantics is generally contrasted with syntax: syntax refers to the surface form of an expression, while the semantics refers to its deeper

meaning in an external World. In computer science, this external World is generally provided by the computational model.

In Computer Science, semantics can be informally defined as the meaning of syntactically correct programs or computable functions. As a simple example, consider two tiny program fragments shown in Equations 4.1 and 4.2. Syntactically, the first statement of each is identical, but the second statements differ. Semantically, however, they are identical: both programs compute the same result, i.e. have the same semantics.

$$x = 1; \quad y = x + x; \tag{4.1}$$

$$x = 1; \quad y = 2 * x; \tag{4.2}$$

Although, the exact definition of semantics for GP is far from obvious, the semantics of an individual is often understood as the behavior of that individual with respect to a set of input values. However the possibilities for computing such semantics depends on the domain. For real valued problems, both canonical-form methods corresponding to Beadle and Johnson's [11] Boolean ROBDDs, and complete enumeration as in McPhee's approach [123], are infeasible. Instead, we propose a simple way to estimate the semantics of subtrees, in which the semantics is approximated by evaluating the subtree on a prespecified set of points in the problem domain. We call this *Sampling Semantics*. Formally, the *Sampling Semantics* of any tree (subtree) is defined as follows:

Definition 1 Let F be a function expressed by a tree (subtree) T on a domain D. Let P be a set of points from domain D, $P = \{p_1, p_2, ...p_N\}$. Then the *Sampling Semantics* of T on P in domain D is the set $S = \{s_1, s_2, ..., s_N\}$ where $s_i = F(p_i), i = 1, 2, .., N$. □

For example, suppose that we are considering the interval [0,1] and using a set of three points, $P = \{0, 0.5, 1\}$, for evaluating semantics. Then, the *Sampling Semantics* of subtree St in Figure 4.1 on P is the set of three values $SS = \{sin(1) - 0, sin(1) - 0.5, sin(1) - 1\}$=$\{0.84, 0.34, -0.16\}$.

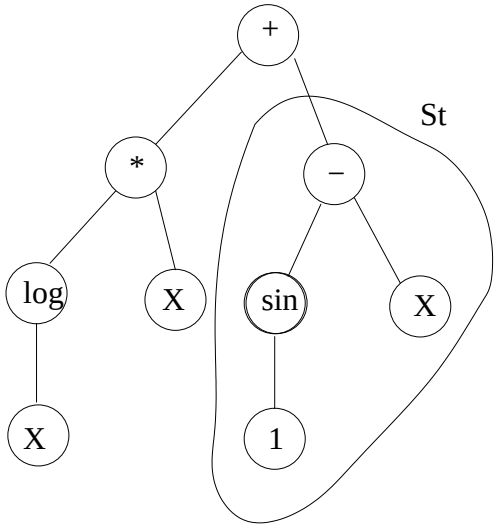

Fig. 4.1: Tree with subtree (for illustrating *Sampling Semantics*).

The value of N depends on the problem. If it is too small, the approximate semantics might be too coarse-grained and not sufficiently accurate. If N is too big, the approximate semantics might be more accurate, but more time consuming to measure. The choice of P is also important. If the members of P are too closely related to the GP function set (for example, π for trigonometric functions, or e for exponential/logarithmic functions), then the semantics might be misleading. For this reason, choosing them randomly may be the best solution. In this thesis, the number of points for evaluating *Sampling Semantics* is often set as the number of fitness cases for the problem, and in the next chapter, we choose the set of points P uniformly randomly from the problem domain. It should be noted that since Sampling Semantics is defined for any subtree, it can be used in particular to estimate the semantics of the whole tree. We will use it in this way in the examples in later sections.

4.1.2 Semantic Distance

Based on the *Sampling Semantics* (SS), we define a *Sampling Semantics Distance* between two subtrees. In our earlier work [130], we defined the *Sampling Semantics Distance* as the sum of the absolute differences for all values of SS. That is, let $P = \{p_1, p_2, ..., p_N\}$

and $Q = \{q_1, q_2, ..., q_N\}$ be the SS of $Subtree_1(St_1)$ and $Subtree_2(St_2)$ on the same set of sample points, then the *Sampling Semantics Distance* (SSD) between St_1 and St_2 was defined as:

$$SSD(St_1, St_2) = |p_1 - q_1| + |p_2 - q_2| + ... + |p_N - q_N| \tag{4.3}$$

While the experiments in [130] showed that this SSD is beneficial, it has the undoubted weakness that the value of the SSD depends on the number of SS points (N). To reduce this drawback, we now use the mean of the absolute differences as the SSD between subtrees. In other word, the SSD between St_1 and St_2 is defined as:

$$SSD(St_1, St_2) = \frac{|p_1 - q_1| + |p_2 - q_2| + ... + |p_N - q_N|}{N} \tag{4.4}$$

4.1.3 Semantic Relationships

Based on *Sampling Semantics Distance*, we can define two semantic relationships between subtrees. The first semantic relationship is *Semantic Equivalence* (SE). Two subtrees are *Semantically Equivalent* on a domain if their SSD on the same sample set of points is sufficiently similar (subject to a parameter called *semantic sensitivity*) – formally:

$$SE(St_1, St_2) = \quad \textbf{if } \mathrm{SSD}(St_1, St_2) < \epsilon \textbf{ then true}$$
$$\textbf{else false}$$

ϵ is the predefined *semantic sensitivity*. This subtree semantic relationship is similar to the metric we used in [130], and was inspired by the work of Mori et al. [127] on GP simplification. The experimental results show that this semantic relationship benefits the GP search process [130, 127].

The second relationship is known as *Semantic Similarity*.[1] The intuition behind semantic similarity is that exchange of subtrees is most likely to be beneficial if the two subtrees are not semantically identical, but also not too semantically dissimilar. Two subtrees St_1 and St_2 are semantically similar on a domain if their SSD on the same sample set lies within a positive interval – formally:

$$SSi(St_1, St_2) = \quad \textbf{if } \alpha < \mathrm{SSD}(St_1, St_2) < \beta \textbf{ then true}$$
$$\textbf{else false}$$

here α and β are two predefined constants, known as the lower and the upper bounds of semantic sensitivity, respectively. Conceivably, the best values for the *lower* and the *upper bound of semantic sensitivity* might be problem dependent. However we suspect that for most symbolic regression problems, there is a wide range of appropriate values (see Chapter 5, where we study various ranges of both the *lower* and the *upper bound of semantic sensitivity*).

We note that there is some biological motivation for this approach. In mammals, the Major Histocompatibility Complex (MHC) genes (on chromosome 6 in humans) play a major role in the immune response, and thus are a key part of our defences against disease, and subject to strong and rapidly-changing evolutionary pressures. However they also play an important role both in mate selection (partners in the same species, but with dissimilar MHC genes, are preferred) [21], and in speciation, because differences in MHC that are too big may cause an immune response from the mother to the foetus [43]. Thus in this case at least, biology also appears to favour crossovers with semantic similarity lying in a specific range.

We conclude this section by highlighting some important differences between our se-

[1] We are using similarity here in its ordinary English meaning, where A is similar to B implies that A is not the same as B, as opposed to a common mathematical convention in which similarity includes equivalence.

Algorithm 1: Semantic Aware Crossover

```
select Parent 1 P_1;
select Parent 2 P_2;
choose at random crossover points at Subtree_1 in P_1;
choose at random crossover points at Subtree_2 in P_2;
generate a number of random points (P) on the problem domain;
calculate the SSD between Subtree_1 and Subtree_2 on P
if Subtree_1 is not semantically equivalent with Subtree_2 then
    execute crossover;
    add the children to the new population;
    return true;
else
    choose at random crossover points at Subtree_1 in P_1;
    choose at random crossover points at Subtree_2 in P_2;
    execute crossover;
    return true;
```

mantic relations and fitness. First, for fitness calculation we need to know the fitness cases, and fitness reflects how good (close to the target function) an individual is. In measuring SS, we do not need to know the fitness cases (of course semantics can be measured using the fitness cases, but different cases can also be used). Second, fitness is measured for the whole individual, while SS is mainly used to encapsulate the semantics of subtrees. The last and most important difference is the objective: fitness is used for individual selection while SS is used to guide operator.

It is also noted that the semantic definition in Krawiec and Lichocki [103] is a particular case of *Sampling Semantics*, in which the set of sample points is the same as the set of fitness cases, and the semantics of the whole tree (a particular subtree) is used in crossover.

4.2 Semantic based Crossovers

Given our new definition of semantics and measures to compare semantics of subtrees, in this section we present novel semantic based crossovers. These crossovers address two main objectives: promoting semantic diversity and improving semantic locality.

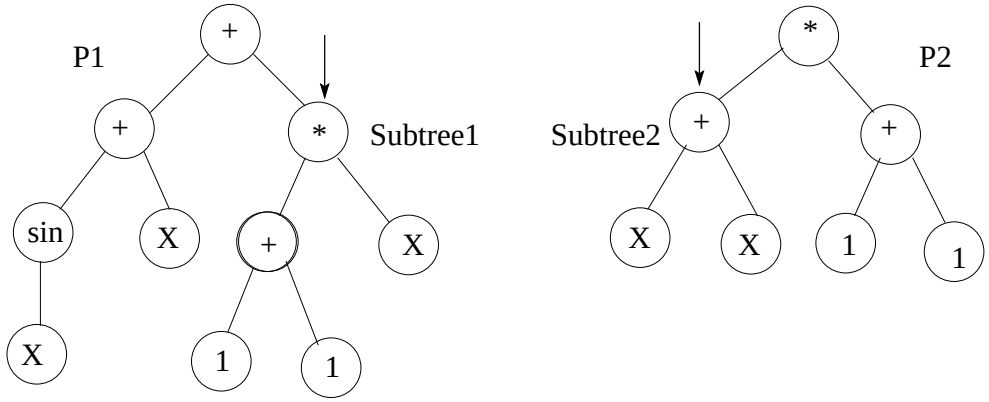

Fig. 4.2: Semantic equivalent subtrees are selected.

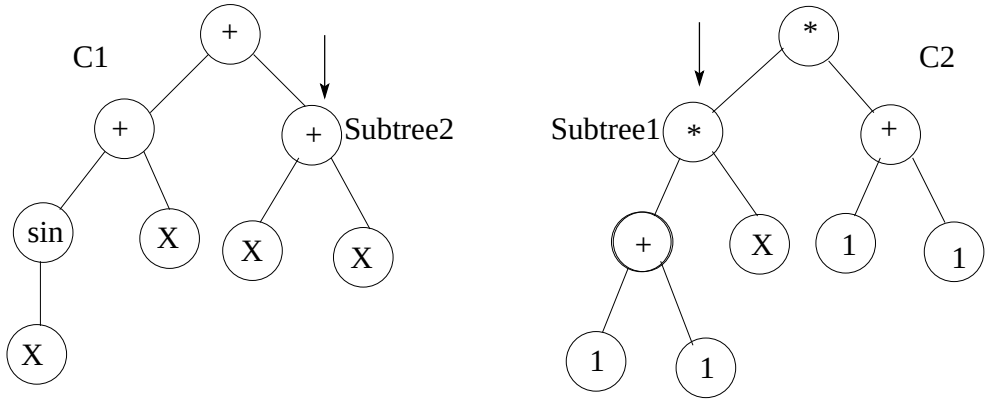

Fig. 4.3: The generated children from semantic equivalent subtree crossover.

4.2.1 Crossover for Promoting Semantic Diversity

As discussed in the previous chapter, promoting semantic diversity plays an important role in the efficiency of the GP search. However, most of the previous work only focused on the GP initialisation phase except two semantic based crossover approaches: Semantic Driven Crossover [86] and No Same Mate selection [62]. In this subsection we give a brief overview of our novel crossover operator for promoting semantic diversity called *Semantic Aware Crossover* (SAC) [130].

SAC is motivated by the observation that GP crossover may exchange semantically equivalent subtrees, resulting in children that are identical to their parents. Consider two selected parents P_1 and P_2 shown at the top of Figure 4.2. P_1 has the semantics

Algorithm 2: Semantic Similarity based Crossover

```
select Parent 1 P_1;
select Parent 2 P_2;
Count=0;
while Count<Max_Trial do
    choose a random crossover point Subtree_1 in P_1;
    choose a random crossover point Subtree_2 in P_2;
    generate a number of random points (P) on the problem domain;
    calculate the SSD between Subtree_1 and Subtree_2 on P
    if Subtree_1 is semantically similar to Subtree_2 then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
choose a random crossover point Subtree_1 in P_1;
choose a random crossover point Subtree_2 in P_2;
execute crossover;
return true;
```

of $sin(X) + 3X$ and P_2 has the semantics of 4X. $Subtree_1$ of P_1 and $Subtree_2$ of P_2 are semantically equivalent subtrees, both having semantics of $2X$, even though their structures are totally different. When these two subtrees are selected for crossover, the children are as shown in Figure 4.3. Obviously, these two children have different syntax (structure) from, but identical semantics to their parents. C_1 has semantics of $sin(X) + 3X$ and C_2 has semantics of $4X$. This leaves the fitness of the children unchanged after crossover.

SAC prevents the swapping of semantically equivalent subtrees in crossover. In other words, each time two subtrees are chosen for crossover, a semantic check (using *Semantic Equivalence*) is performed to determine if they are equivalent. If they are, the crossover is aborted and instead performed on two other randomly chosen subtrees. The details of SAC's algorithm is given in Algorithm 1.

4.2.2 Crossover for Improving Semantic Locality

Previous research [69, 171] showed the crucial role of locality (of representation and operators) in the field of EC in general and GP in particular. However, they purely focused on the locality in terms of the syntactical aspect of representations and operators. In this subsection, we introduce a new crossover, Semantic Similarity based Crossover (SSC), that achieves higher locality in terms of semantics than standard crossover.

The semantic based crossover to improve semantic locality, SSC, is an extension of SAC in two ways. First, when two subtrees are selected for crossover, their semantic similarity, rather than semantic equivalence, is checked. Since semantic similarity is usually more difficult to satisfy than semantic equivalence, repeated failures may occur. Therefore, SSC uses multiple trials to find semantically similar pairs, only reverting to random selection after passing a bound on the number of trials. The number of trials to find a semantically similar pair is called *Max_Trial* (MT). Algorithm 2 shows how SSC works in detail.

In our experiments, we set MT to several different values to see how these values affect SSC. The motivation for SSC is to encourage exchange of semantically different, but not substantially different, subtrees. In other word, while forcing a change in the semantics of the individuals in the population, we want to keep this change bounded and small. We anticipate that a smoother change in semantics of the individuals will result, and might lead to a smoother change in fitness of the individuals after crossover.

For instance, consider two parents selected for crossover in Figure 4.4. Assume that we measure the SS of a tree on 10 points, $P = \{1, 2, ..., 10\}$. Then the SS of parents P_1, P_2 and of Subtree1 (St_1), Subtree2 (St_2), and Subtree3 (St_3) are as shown in Table 4.1 and Table 4.2. It can be seen from these tables that St_1 and St_2 are semantically similar (using α=10^{-3}, β=0.4) as their SSD is 0.09 and St_1 and St_3 are semantically dissimilar since the SSD is 4.5. If crossover is performed by swapping two semantically similar subtrees (St_1 and St_2), the generated children are show in Figure 4.5. The SS of the two children (C_1, C_2) are shown in Table 4.1. We can also measure the SSD between C_1 and P_1 and between C_2 and P_2 (as shown in columns $C_1 - P_1$ and $C_2 - P_2$ in Table 4.1). Evidently, the change

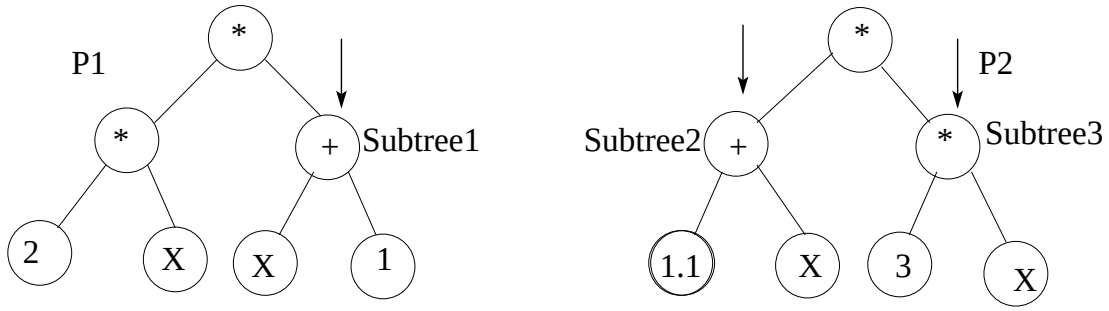

Fig. 4.4: Parents for crossover.

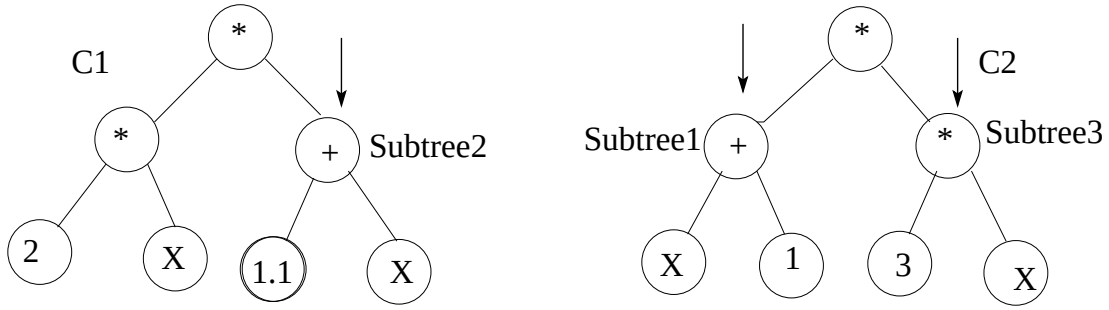

Fig. 4.5: Children generated by crossing over two semantically similar subtrees.

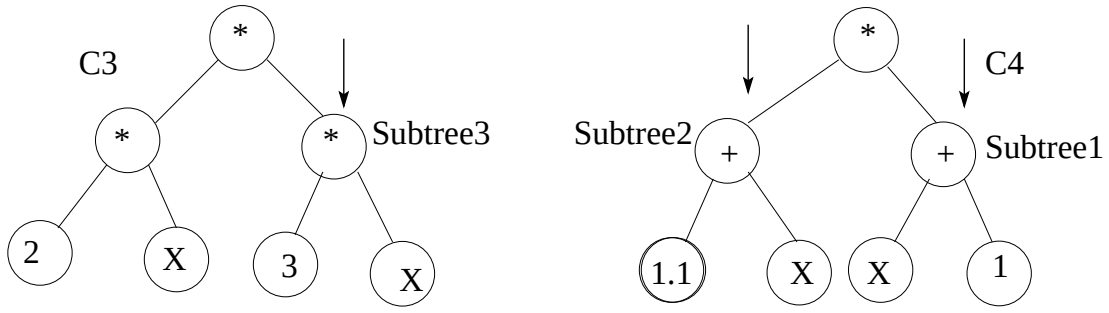

Fig. 4.6: Children generated by crossing over two semantically dissimilar subtrees.

Tab. 4.1: Sampling semantics of parents, subtrees and children when swapping two similar subtrees.

Points	P_1	P_2	St_1	St_2	St_1-St_2	C_1	C_2	C_1-P_1	C_2-P_2
1	4	6.3	2	2.1	0.1	4.2	6	0.2	0.3
2	12	18.6	3	3.1	0.1	12.4	18	0.4	0.6
3	24	36.9	4	4.1	0.1	24.6	36	0.6	0.9
4	40	61.2	5	5.1	0.1	40.8	61	0.8	1.2
5	60	91.5	6	6.1	0.1	61.0	91	1.0	1.5
6	84	127.8	7	7.1	0.1	85.2	127	1.2	1.8
7	112	170.1	8	8.1	0.1	113.4	170	1.4	2.1
8	144	218.4	9	9.1	0.1	145.6	218	1.6	2.4
9	180	272.7	10	10.1	0.1	181.8	272	1.8	2.7
10	220	333.0	11	11.1	0.1	222.0	333	2.0	3.0
SSD					0.09			1.1	1.65

Tab. 4.2: Sampling semantics of parents, subtrees and children when swapping two dissimilar subtrees.

Points	P_1	P_2	St_1	St_3	St_1-St_3	C_3	C_4	C_3-P_1	C_4-P_2
1	4	6.3	2	2	0	6	4.2	2	2.1
2	12	18.6	3	4	1	24	9.3	12	9.3
3	24	36.9	4	6	2	54	16.4	30	20.5
4	40	61.2	5	8	3	96	25.5	56	35.7
5	60	91.5	6	10	4	150	36.6	90	54.9
6	84	127.8	7	12	5	216	49.7	132	78.1
7	112	170.1	8	14	6	294	64.8	182	105.3
8	144	218.4	9	16	7	384	81.8	240	136.5
9	180	272.7	10	18	8	486	101.0	306	171.7
10	220	333.0	11	20	9	600	122.1	380	201.0
TSD					4.5			143	82.5

Algorithm 3: Semantic Aware Mutation

```
select Parent 1 P;
choose at random mutation points at Subtree_1 in P;
generate at random a subtree Subtree_2;
generate a number of random points (Q) on the problem domain;
calculate the SSD between Subtree_1 and Subtree_2 on Q
if Subtree_1 is not semantically equivalent with Subtree_2 then
    execute mutation by replacing Subtree_1 with Subtree_2;
    add the child to the new population;
    return true;
else
    choose at random mutation points at Subtree_1 in P;
    generate at random a subtree Subtree_2;
    execute mutation by replacing Subtree_1 with Subtree_2;
    add the child to the new population;
    return true;
```

of semantics through crossover is quite small (1.1 with C1 and 1.65 with C2). This, we hope, will help to make a smoother change of fitness.

By contrast, if crossover is conducted by swapping two dissimilar subtrees (St_1 and St_3), the children are shown in Figure 4.6. The results of the calculation of the SS of the two children (C_3 and C_4) and the semantic distances between these children and their parents are shown in Table 4.2. It can be seen from this table that the change in semantics between parents and children is rather large (143 and 82.5 for $C3$ and $C4$, respectively). This, we anticipate, will lead to an abrupt change in fitness after crossover.

4.3 Semantic based Mutations

This section presents two new semantic based mutations. These mutations also address two main objectives: promoting semantic diversity and improving semantic locality.

Algorithm 4: Semantic Similarity based Mutation

```
select a Parent P;
Count=0;
while Count<Max_Trial do
    choose a random mutation point Subtree_1 in P;
    generate at random a subtree Subtree_2;
    generate a number of random points (Q) on the problem domain;
    calculate the SSD between Subtree_1 and Subtree_2 on Q
    if Subtree_1 is semantically similar to Subtree_2 then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
choose a random mutation point Subtree_1 in P;
generate at random a subtree Subtree_2;
execute crossover;
return true;
```

4.3.1 Mutation for Promoting Semantic Diversity

The mutation for promoting semantic diversity is inspired from the crossover for promoting semantic diversity, SAC. This mutation is called *Semantic Aware Mutation* (SAM). SAM is performed in a similar way with SAC, but adapted to constrain the semantics in mutation rather than in crossover. In other words, a parent is selected for mutation by applying one of the selection approaches. A mutation point is randomly chosen in the parent and a new subtree is stochastically generated. Then the semantic equivalence is checked to determine if these two subtrees (replaced and replacing subtrees in the mutation operation) are equivalent. If they are not semantically equivalent, the mutation is performed by simply replacing the subtree at the mutation point with the new generated subtree. On the other hand, if they are equivalent then, the mutation is redone by randomly selecting another mutation point and stochastically generating a new subtree. Algorithm 3 shows how SAM works in detail.

4.3.2 Mutation for Improving Semantic Locality

The mutation for improving semantic locality is inspired from the crossover for improving semantic locality, SSC. This mutation is called *Semantic Similarity-based Mutation* (SSM). SSM is implemented in the similar to SSC, but adapted to constrain the semantics in mutation rather than in crossover. Similar to SAM, a parent is selected for mutation by applying one of the selection approaches. A mutation point is randomly chosen in the parent and a new subtree is stochastically generated. Then the semantic similarity is checked to determine if these two subtrees (replaced and replacing subtrees in the mutation operation) are semantically similar. If they are similar, mutation is performed by simply replacing the subtree at the mutation point with the new generated subtree. If they are not semantically similar, SSM repeats randomly selecting another mutation point and stochastically generating a new subtree. This process is continued until we can find a semantically similar subtree pair or pass a number of trials. In this cases, SSM is executed by reverting to standard mutation. Algorithm 4 shows how SSM works in detail.

4.4 Conclusion

This chapter presented the foundation for the thesis. A method that measured the semantics of a subtree in GP was proposed. This semantics of a subtree is called *Sampling Semantics*. Sampling Semantics of a subtree is measured by evaluating the subtree on a number of points sampled from a problem domain. From this, a semantic distance and two semantic relationships are defined. Next, a number of semantic based operators were introduced. These operators includes crossover for promoting semantic diversity, crossover for improving semantic locality, mutation to promote semantic diversity and mutation to improve semantic locality. The impact of these semantic based operators on GP behaviour will be investigated in the following chapter.

Chapter 5

Semantic based Operators: Comparative Results

The previous chapter proposed some novel semantic based operators. This chapter presents comparative results of semantic based operators and standard operators. The experiment settings are described first. After that, the detailed experimental results of the semantic based crossovers and standard crossover are given. Next are the comparison between semantic based mutations and standard mutation. Then, we combine both semantic based crossover and semantic based mutation and contrast them with a GP system using standard crossover and standard mutation. Finally, an analysis of the sensitivity of the parameters of these semantic based operators is discussed. The results in this chapter have been published in [130, 132, 131, 140].

5.1 Experimental Settings

This section presents a family of problems that will be used for testing the semantic based operators proposed in the previous chapter. The parameter settings for the GP systems in our experiments are also given.

Tab. 5.1: Symbolic Regression Functions.

Functions	Fitcases
$F_1 = x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_2 = x^5 + x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_3 = sin(x^2)cos(x) - 1$	20 random points $\subseteq$ [-1,1]
$F_4 = sin(x) + sin(x + x^2)$	20 random points $\subseteq$ [-1,1]
$F_5 = log(x + 1) + log(x^2 + 1)$	20 random points $\subseteq$ [0,2]
$F_6 = \sqrt{x}$	20 random points $\subseteq$ [0,4]
$F_7 = sin(x) + sin(y^2)$	100 random points $\subseteq$ [0,1]x[0,1]
$F_8 = 2sin(x)cos(y)$	100 random points $\subseteq$ [0,1]x[0,1]

5.1.1 Symbolic Regression Problems

To test the performance of semantic based operators in comparison to the standard GP operators, eight real valued symbolic regression problems are employed as a test suite. These problems are grouped into three categories: learning polynomial functions; trigonometric, logarithm and square-root functions; and bivariate functions. Most of them are taken from the works of Hoai et al. [68], Keijzer [92].

The task of training GP to learn a function in symbolic form is to teach GP to search a solution model that fits a given finite sample of data. This finite sample of data is called the training set. The training sets of the functions in the test suite include 20 random points for single variable functions and 100 random points for bivariate functions from the intervals of interest. These functions and their training sets are shown in Table 5.1

5.1.2 Parameter Settings

In all systems, GP with standard operators and GP with semantic based operators were tried on the eight real valued symbolic regression problems above. All systems used the same random number generators with the same set of seeds and were run on an identical platform. Parameter settings for all systems are typical settings that have been used in previous work [98, 68]. They are listed as follows: population size - POPSIZE=500, maximal number of generations - MAXGEN=50, selection mechanism = tournament selection,

Tab. 5.2: Run and Evolutionary Parameter Values.

Parameters	Value
Population size	500
Generation	50
Selection	Tournament
Tournament size	3
Initial Max depth	6
Max depth	15
Max depth of mutation tree	5
Non-terminals	+, -, *, / (protected versions), sin, cos, exp, log (protected versions)
Terminals	X, 1 for single variable problems, and X,Y for bivariable problems
Raw fitness	mean of absolute error on all fitness cases
Hit	when an individual has an absolute error < 0.01 on a fitness case
Successful run	when an individual scores hits on all fitness cases
Trials per treatment	100 independent runs for each value

tournament size of selection = 3, the max depth of initial trees - MAXINITDEPTH = 6, the max depth of trees over the course of evolution - MAXDEPTH = 15, the max depth of mutation tree - MAXMUTDEPTH = 5. The initialisation method used for GP was ramped-half-and-half.

The terminal set includes two sets. The first set is {X, 1} for single variable functions and the second one contains {X, Y} for bivariate functions. The function set are $F = \{+, -, *, /, sin, cos, exp, log\}$. 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 base 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 *raw fitness function* is the mean of the absolute error over all fitness cases. A hit is counted when an individual has an absolute error less than 0.01 on a fitness case and a run is considered as *successful* when some individual hits (i.e. absolute error <0.01) every fitness case. For each set of parameter settings, 100 independent runs were performed. The basic parameters settings are given in Table 5.2. Notice that the crossover and mutation rates are not fixed. They will be set up according to the set of experiments.

5.2 A Comparative Result of Crossovers

This section presents the results from a comparison of GP with semantic based crossovers and standard crossover. The experiment settings for GP systems are given in Table 5.2. The crossover rate was set at 0.9 and the mutation rate was 0.05. Although these experiments purely concern crossover, we have retained mutation at a low rate with an aim to study crossover in the context of a normal GP run.

The semantic sensitivities for Semantic Aware Crossover (SAC) were set as 10^{-X} with X=2, 3, 4. They are values for the good performance of SAC. It forms three tested configurations of SAC called as SACX with X=2, 3, 4. For Semantic Similarity based Crossover (SSC), the lower bound of semantic sensitivity is fixed at 10^{-3}, and three values of the upper bound: 0.2, 0.3, and 0.4 are used. The value of Max_Trial of SSC is 12. This value was calibrated from our experiments as one of the best values for SSC. Three experiment configurations of SSC will be called SSCX with X = 02, 03, 04 respectively.

5.2.1 Results

For all crossovers, two classic performance metrics, namely mean best fitness and the number of successful runs are recorded. The results of the number of successful runs (out of 100 runs) of these crossovers are presented in Table 5.3. Table 5.4 shows the best fitness found, averaged over all 100 runs of each GP system. We tested the statistical significance of the results in Table 5.4 using a Wilcoxon signed-rank test with a confidence level of

Tab. 5.3: Number of successful runs out of 100 runs of three crossovers.

Crossovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	12	4	40	3	20	17	31	18
SAC2	18	5	42	9	22	18	29	16
SAC3	13	6	45	9	26	19	25	17
SAC4	12	6	46	6	23	19	25	16
SSC02	20	9	84	26	55	34	77	42
SSC03	26	8	83	18	53	40	66	42
SSC04	25	15	71	22	55	50	43	26

Tab. 5.4: Mean best fitness of three crossovers. Note that the values are scaled by 10^2.

Crossovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.54	2.06	0.69	1.43	0.86	1.28	1.68	1.21
SAC2	**1.32**	**1.73**	0.67	1.41	0.84	1.18	1.66	1.41
SAC3	1.49	1.75	0.66	1.29	0.82	1.24	1.87	1.33
SAC4	1.45	1.74	0.66	1.26	0.83	1.26	1.87	1.35
SSC02	**0.95**	**1.27**	**0.21**	**0.74**	**0.39**	**0.61**	**0.43**	**0.62**
SSC03	**0.87**	**1.06**	**0.19**	**0.80**	**0.39**	**0.66**	**0.50**	**0.64**
SSC04	**0.94**	**1.05**	**0.31**	**0.83**	**0.35**	**0.59**	**0.85**	**0.80**

95%. In Table 5.4, if a run is significantly better than Standard Crossover (SC), its result is printed bold face.

5.2.2 Discussion of Crossover Results

We first look at the effect of promoting semantic diversity (via comparison between SC and SAC). Unsurprisingly, promoting semantic diversity brings some positive effects to the performance of GP. This is confirmed by the results in both Table 5.3 and Table 5.4. It can be seen from Table 5.3 that the crossover, SAC, that promotes semantic diversity found solutions more easily on some problems. This is reflected by a greater number of solutions found by semantic based crossover for enhancing diversity over SC. However it seems that the positive effects of SAC is not universal. SAC gives no improvement on bivariate functions (F_7 and F_8).

Tab. 5.5: Average time of a run in milliseconds of three crossovers.

Crossovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	3042	3092	3112	4022	3428	3646	10180	10128
SAC2	3551	3428	3316	4260	3505	3845	11557	12234
SAC3	3403	3373	3265	4158	3435	3778	11950	11874
SAC4	3526	3483	3374	4165	3474	3665	11399	11936
SSC02	3388	3382	3649	4697	4470	4106	11252	12466
SSC03	3802	3645	3937	4935	4410	4356	12709	12879
SSC04	3899	3964	3713	4668	4690	4542	12621	12775

The mean best fitnesses in Table 5.4 are consistent with this, in that promoting semantic diversity brings a positive influence to the quality of solutions in some situations. The mean best fitness of the crossover which promotes semantic diversity is usually smaller than that of SC. However, a few of the differences are significant. These results are not entirely surprising as we will see in the latter chapter that even SC can achieve quite high semantic diversity on these real valued problems. Usually, from 60% to 70% of SC operations actually change the semantics between parents and children. This is much higher than for Boolean problems, where 60% to 70% of SC crossovers do not create different semantics in the children from the parents [123]. Generally, promoting semantic diversity of crossover leads to a positive effect on the performance of GP. It is generally not, however, a substantial or consistent contributor to increased performance on real valued problems.

On the other hand, improving semantic locality helps to further enhance the performance of GP in comparison with semantic diversity promotion alone. The results in both tables show how effective the semantic locality of crossover is at contributing to GP performance. It can be observed from Table 5.3 that the version of crossover that keeps a semantically small change (i.e. SSC) leads to further improvements of the GP system in comparison with the version that only promotes semantic diversity. In general, SSC is much better than both SC and the crossover for promoting semantic diversity, SAC, in terms of finding solutions.

Table 5.4 is consistent with Table 5.3, as it confirms that the performance of GP can be

enhanced by improving locality in crossover. It can be seen that controlling locality in SSC leads to a consistently significant improvement over SC in comparison with approaches that just promote diversity such as SAC. The Wilcoxon signed-rank test results show that only two cases of the improvement of SAC over SC is significant, by contrast, SSC is always significantly better than SC on all problems with all tested values of the upper bound of semantic sensitivity.

However, the advantage of semantic based crossovers comes at a cost, it takes time to check semantics before crossing over. To estimate the extra time of the semantic checking of these crossovers, we measured their running time. The average time of a run of these operators compared to SC is shown in Table 5.5. It can be seen from this table that the average time of a run of both semantic based crossovers, SAC and SSC, are often higher than of SC. It is understandable since both these crossovers need extra time to evaluate semantics of subtrees. However, the overhead for both SAC and SSC is not large. The table also shows that the average running time of SSC is only slightly higher than that of SAC although SSC repeats more trial times to select two semantic similar subtrees. The results in Chapter 7 will show that while promoting semantic diversity of a crossover often leads to slightly more bloat than standard crossover, improving semantic locality often helps to reduce code bloat. Therefore, SSC does not run much more slowly than SAC eventhough it needs more semantic checking. Moreover, the next chapter will propose an improvement of SSC that helps to speed up semantic checking and it helps SSC to run at the same speed as SC.

In conclusion, the results in this section demonstrate that promoting semantic diversity is necessary but improving semantic locality is even more important. It is strongly believed that enhancing semantical diversity should always be combined with improving semantic locality to gain further improvements of the performance of GP.

Tab. 5.6: Number of successful runs out of 100 runs of three mutations.

Mutations	F1	F2	F3	F4	F5	F6	F7	F8
SM	3	0	14	2	9	4	9	4
SAM2	5	2	19	5	12	5	9	4
SAM3	2	0	20	2	11	3	7	4
SAM4	0	0	15	2	11	5	8	4
SSM02	5	2	50	9	22	15	49	12
SSM03	9	1	59	9	19	9	49	16
SSM04	10	3	51	5	18	14	41	15

5.3 A Comparative Result of Mutation

The previous section shows that semantic based crossovers, especially SSC, often help to improve the performance of GP compared to standard crossover. This raises the question if semantics also plays a similarly important role with mutation. This section presents the comparative results between semantic based mutations and standard mutation. The basic experiment settings for GP systems are given in Table 5.2. The crossover rate was set at 0.05 and the mutation rate was set at 0.9. The configuration and naming convention for Semantic Aware Mutation (SAM) and Semantic Similarity base Mutation (SSM) are similar to SAC and SSC in the previous section, respectively.

5.3.1 Results

Similar to crossover, for all mutations, two classic performance metrics, namely mean best fitness and the number of successful runs are recorded. The results of the number of successful runs (out of 100 runs) and the average of the best fitness found are shown in Table 5.6 and Table 5.7, respectively. We also tested the statistical significance of the results in Table 5.7 using a Wilcoxon signed-rank test with a confidence level of 95% and if a run is significantly better than SM, its result is printed bold face.

Tab. 5.7: Mean best fitness of three mutations. Note that the values are scaled by 10^2.

Mutations	F1	F2	F3	F4	F5	F6	F7	F8
SM	2.14	2.50	0.95	1.60	1.06	1.56	2.99	2.07
SAM2	2.20	2.68	0.87	1.59	1.04	1.56	2.93	2.24
SAM3	2.16	2.47	0.88	1.54	1.05	1.56	2.88	2.00
SAM4	2.21	2.68	1.05	1.47	1.04	1.49	2.91	2.05
SSM02	**1.37**	**1.73**	**0.45**	**1.07**	**0.74**	**1.20**	**1.30**	**1.39**
SSM03	**1.31**	**1.54**	**0.40**	**1.02**	**0.75**	**1.17**	**0.97**	**0.96**
SSM04	**1.13**	**1.36**	**0.42**	**1.16**	**0.72**	**1.16**	**1.03**	**1.01**

Tab. 5.8: Average time of a run in milliseconds of three mutations.

Mutations	F1	F2	F3	F4	F5	F6	F7	F8
SM	5443	6319	6354	6189	5312	5492	16786	16493
SAM2	5889	6705	6847	6528	5574	5714	18284	18109
SAM3	5783	6773	6676	6536	5656	5847	17031	18756
SAM4	5870	6846	6789	6678	7011	5962	18226	17821
SSM02	8324	8658	8389	8921	8267	8732	21354	22342
SSM02	8427	8781	8478	8965	8221	8789	22313	21324
SSM04	8610	8493	8382	9396	8556	8623	20663	20120

5.3.2 Discussion of Mutation Results

It can be seen that the results of semantic based mutations are similar to crossover, meaning that promoting semantic diversity of mutation brings limited positive effects to the performance of GP while improving semantic locality of mutation helps to further enhance the performance of GP. Table 5.6 shows that the number of successful runs of SAM is often greater than SM and Table 5.7 shows that SAM often found solutions with better quality. However, the signed-rank test results show none of the improvements of SAM over SM are significant.

Conversely, it can be observed from Table 5.6 that the version of mutations that keeps a semantically small change leads to further improvements of the GP system in comparison with the version that purely promotes semantic diversity. In general, SSM is much better than both SM and SAM, in terms of finding solutions. Table 5.7 is consistent with this,

it again confirms that the performance of GP can be enhanced by improving locality in mutations. It can be seen that the mean best fitness of SSM is always smaller than for both SM and SAM. The Wilcoxon signed-rank test results show that SSM is always significantly better than SM on all problems with all tested values of the upper bound of semantic sensitivities.

To estimate the effectiveness of these mutations, we again measured the running time of the three mutations. The average time of a run of these mutations is shown in Table 5.8. Similar to semantic based crossovers, the average time of a run of both semantic-based mutations, SAM and SSM, are often higher than one of SM. However, the overhead of running time of both SAM and SSM is also not large. The table also shows that the extra time of SSM versus SM and SAM is slightly higher than the extra time of SSC versus SC and SAC (See Section 5.2).

Overall, the results in this section demonstrate that semantics also plays an important role in mutation and that promoting semantic diversity of mutations is necessary but improving semantic locality is even more important. Therefore, enhancing semantic diversity of a mutation should always be accompanied with improving semantic locality to gain further improvements of GP performance.

5.4 Promoting Semantic Similarity of both Crossover and Mutation

The previous sections show that improving semantic locality of crossover or mutation helps to improve the performance of GP. The question raised from those results is if simultaneously improving semantic locality of crossover and mutation helps to further enhance GP performance. This section is dedicated to answer that question.

The basic experiment settings for GP systems are given in Table 5.2. The crossover rate was set at 0.9 and the mutation rate was set at 0.9. The GP configuration that uses standard crossover and standard mutation is called SCM. The experimental settings for

SSC and SSM are similar to the previous sections.

The method that improves semantic locality of both crossover and mutation is shorthanded as SSCM. The lower semantic sensitivity for SSCM is also fixed at 10^{-3}, and three values of higher semantic sensitivity: 0.2, 0.3, and 0.4 are used. The value of Max_Trial of SSCM is also set to 12. Three instances of SSCM will be referred as SSCMX with X= 02, 03, 04.

5.4.1 Results

In this experiment, two classic performance metrics, namely mean best fitness and the number of successful runs are recorded. The results of the number of successful runs (out of 100 runs) and the best fitness found, averaged over all 100 runs are shown in Table 5.9 and Table 5.10. We tested the statistical significance of the results in Table 5.10 using a Wilcoxon signed-rank test with a confidence level of 95%. The results in Table 5.10 are printed as follows:

1. If a run of SSC, SSM, and SSCM is not significantly better than SCM, it is printed in plain face

2. If a run of SSC, SSM, and SSCM is significantly better than SCM, it is printed *italic* face

3. If a run of SSCM is significantly better than SCM, SSC and SSM it is printed ***bold and italic*** face**.**

5.4.2 Discussion

The results in both tables show that improving semantic locality of crossover and mutation helps to improve the performance of GP with this configuration. Table 5.9 shows that the crossover based on improving semantic locality, SSC, often finds solutions easier than standard crossover. The exception only lies in one case on function F_1 and one case on

Tab. 5.9: Number of successful runs out of 100 runs when improving semantic locality of both crossover and mutation.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SCM	5	0	11	2	8	10	7	7
SSC02	3	1	30	4	16	13	19	9
SSC03	6	1	38	10	12	8	14	11
SSC04	13	3	29	3	12	12	11	9
SSM02	11	2	35	5	10	4	23	9
SSM03	11	2	33	6	7	6	25	14
SSM04	8	0	25	10	11	14	26	12
SSCM02	10	3	45	8	8	12	41	22
SSCM03	18	7	45	7	13	9	36	25
SSCM04	20	3	56	4	10	13	30	12

function F_6. Similarly, the table also shows that the number of successful runs found by using the mutation, that is improved semantic locality, SSM, is usually greater than those found by SCM. The exception falls in some cases on function F_5 and F_6.

What is a more important result from Table 5.9 is that by improving both semantic locality of crossover and mutation, we can improve the performance of GP to a greater extent. The table shows that the number of successful runs found by SSCM is often greater than ones found by SSC or SSM.

The results in Table 5.10 are consistent with those in Table 5.9 in that it shows the benefit of improving both semantic locality of crossover and mutation. Definitely, SSCM finds solutions with better quality in terms of the best fitness in comparison with SSC and SSM. The results of a Wilcoxon signed-rank test also show that simultaneously improving semantic locality of crossover and mutation often leads to significantly improved GP performance compared to only improving semantic locality of crossover or mutation alone.

Overall, the results in this section and the previous sections show that improving semantic locality of crossover or mutation plays an important role in GP performance. It often leads to significantly enhance the performance of GP. It also shows that when using a high rate of both crossover and mutation, improving semantic locality of crossover and mutation at the same time is better than purely improving semantic locality of only

Tab. 5.10: Mean best fitness when improving semantic locality of both crossover and mutation. Note that the values are scaled by 10^2.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SCM	2.26	3.13	1.01	1.84	1.07	1.94	3.43	2.55
SSC02	*1.75*	*2.39*	*0.72*	*1.39*	0.91	*1.44*	*2.56*	*1.94*
SSC03	*1.60*	*2.27*	*0.68*	*1.34*	0.86	*1.68*	*2.40*	*1.66*
SSC04	*1.59*	*2.13*	*0.66*	*1.46*	0.98	*1.42*	*2.77*	*1.95*
SSM02	*1.45*	*2.24*	*0.61*	*1.51*	1.17	*1.47*	*2.01*	*1.91*
SSM03	*1.45*	*2.03*	*0.63*	*1.30*	1.06	*1.39*	*1.92*	*1.54*
SSM04	*1.53*	*2.06*	*0.63*	*1.39*	0.87	*1.45*	*1.67*	*1.71*
SSCM02	*1.46*	***1.72***	*0.52*	***1.17***	0.89	*1.26*	***1.13***	***1.31***
SSCM03	***1.15***	***1.61***	***0.45***	***1.16***	0.89	***1.12***	***1.11***	***1.09***
SSCM04	***1.09***	***1.59***	***0.39***	***1.09***	*0.80*	*1.23*	***1.09***	***1.12***

crossover or mutation.

5.5 Parameters Sensitivity Analysis

The preceding sections showed that crossover and mutation that improve semantic locality often result in a significant improvement of GP performance. However, how sensitive are these operators to their parameter settings? The experiments in this sections investigate the effect of changing some parameters on the performance of semantic similarity based crossover and semantic similarity based mutation. An analysis of parameters sensitivity of SSC is presented first. After that, the effect of changing parameters to SSM is given and discussed.

5.5.1 Parameters Sensitivity Analysis of Semantic Similarity based Crossover

The previous sections showed the benefit of using semantic similarity based crossover, SSC. However, the performance of SSC could also depend on the selection of some parameters. The incorrect setting of these parameters could lead to a poor performance of SSC. This

subsection is dedicated to examine how SSC's parameters affect its performance. The GP parameters were setup as in Table 5.2. The crossover rate was set at 0.9 and mutation rate was set at 0.05.

Three parameters of SSC were investigated, namely, the *lower bound of semantic sensitivity* (LBSS), the *upper bound of semantic sensitivity* (UBSS), Max_Trial (MT) used to selection two semantically similar subtrees. First, we examined the effect of the most important parameter, UBSS. We fixed the other parameters as follows: LBSS: 10^{-3}, MT: 12. The UBSS was set at 6 values: 0.1, 0.2, 0.4, 0.6, 0.8, and 1. Six these configurations of SSC are denoted as SSCUX where X is 01, 02, 04, 06, 08, or 1.

The second experiment analysed the effect of LBSS. In this experiment, the other parameters were set as follows: UBSS= 0.4, MT= 12. Five values for LBSS were investigated, i.e. 10^{-X} where (X=1, 2, 3, 4, and 5). These five instances of SSC are referred as SSCLX with X=1, 2, 3, 4, and 5.

The last experiment tested sensitivity to the number of trials allowed in selecting semantically similar subtrees in SSC (MT). For this experiment, LBSS= 10^{-3}, UBSS=0.4. MT was set at 4, 8, 12, 16, 20, 24. These configurations of SSC are denoted as SSCMTX, with X=4, 8, 12, 16, and 20, 24.

To estimate the effect of changing these parameters, we recorded the best fitness of a run. These values were averaged over 100 runs and are shown in Table 5.11. For the purpose of comparison, the mean best fitness of standard crossover is also shown in the top row of this table.

We can see that the value of UBSS has a remarkable effect on the performance of SSC. It seems that values from 0.2 to 0.8 are suitable for the problems under test, with values from 0.2 to 0.6 being the best. If UBSS is too small (0.1) or too big (1) the performance of SSC is poorer. This can be explained by recording the percentage of SSC that successfully selects two semantically similar subtrees, as shown in Table 5.12. The values for SSCLX are not shown in this table as they have little effect. We can see that if UBSS is too small, only a few SSCs can succeed in exchanging semantically similar subtrees (from 30% to 40% when UBSS is 0.1), so that SSC underperforms. We have tried increasing the Max_Trial

Tab. 5.11: Mean best fitness of SSC with different parameters. Note that the values are scaled by 10^2.

Crossovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.54	2.02	0.60	1.43	0.76	1.18	1.68	1.11
SSCU01	0.99	1.38	0.38	0.91	0.51	0.99	0.85	0.82
SSCU02	0.95	1.27	0.31	0.84	0.39	0.61	0.63	0.72
SSCU04	0.94	1.05	0.21	0.73	0.35	0.59	0.85	0.80
SSCU06	0.94	1.10	0.24	0.92	0.40	0.73	1.03	0.92
SSCU08	0.97	1.11	0.37	0.97	0.47	0.57	1.20	1.10
SSCU1	1.00	1.45	0.40	1.01	0.52	0.75	1.31	1.20
SSCL1	1.04	1.28	0.45	0.97	0.55	0.79	1.50	1.02
SSCL2	0.92	0.97	0.25	0.82	0.40	0.66	0.83	0.88
SSCL3	0.94	1.05	0.31	0.83	0.35	0.59	0.85	0.80
SSCL4	0.74	1.06	0.25	0.79	0.46	0.55	0.84	0.83
SSCL5	0.78	1.09	0.25	0.78	0.43	0.57	0.86	0.79
SSCMT4	1.00	1.32	0.32	1.06	0.51	0.78	1.11	0.83
SSCMT8	0.74	1.09	0.31	0.77	0.45	0.64	0.98	0.87
SSCMT12	0.94	1.05	0.31	0.83	0.35	0.59	0.85	0.80
SSCMT16	0.86	0.98	0.21	0.78	0.42	0.59	0.77	0.87
SSCMT20	0.90	1.08	0.33	0.82	0.41	0.52	0.82	0.75
SSCMT24	0.73	1.02	0.20	0.89	0.38	0.48	0.91	0.84

to compensate for decreasing the upper bound. This was unsuccessful, as if UBSS is too small, the exchange of semantics between the two parents is also too small, so that SSC is more readily trapped in local optima. By contrast, if UBSS is too large, it is almost trivial to find semantically similar subtrees (almost 100% for UBSS=1) because most subtrees are sufficiently semantically similar, so that SSC behaves like SC.

While changing UBSS has a remarkable effect on SSC, LBSS has almost no effect on performance provided it is sufficiently small. Table 5.11 shows that while LBSS was set to small values (from 10^{-2} to 10^{-5}), the performance of SSC was almost unchanged. In order to understand this, we recorded the percentage of subtrees with SSD smaller than 10^{-2} that are actually semantically identical. In fact, 99% of such semantically equivalent subtrees actually have the same semantics. Thus 99% of these subtrees would have satisfied the equivalence condition regardless of the values of LBSS. Only in the case when LBSS

Tab. 5.12: The percentage of SSC that successfully exchange two semantically similar subtrees.

Crossovers	F1	F2	F3	F4	F5	F6	F7	F8
SSCU01	43.4	42.9	51.1	25.8	28.7	19.8	58.2	54.5
SSCU02	66.2	62.1	76.0	53.9	57.0	43.6	73.8	70.0
SSCU04	80.6	79.3	90.6	89.8	82.9	64.4	92.2	93.8
SSCU06	95.0	95.2	97.2	98.5	91.6	81.1	98.1	97.3
SSCU08	98.5	98.3	98.0	99.4	95.3	89.5	99.8	99.8
SSCU1	99.7	99.7	98.2	99.8	98.3	94.9	99.8	99.8
SSCMT4	41.0	41.3	61.7	54.0	46.8	25.8	61.2	59.9
SSCMT8	67.1	66.5	81.6	78.9	72.9	50.3	84.6	85.2
SSCMT12	80.6	79.3	90.6	89.8	82.9	64.4	92.2	93.8
SSCMT16	86.6	86.0	94.1	93.2	87.9	73.0	94.2	94.6
SSCMT20	91.8	90.9	95.2	96.7	91.6	78.8	96.3	96.5
SSCMT24	93.7	92.8	97.2	98.1	93.8	82.1	97.8	97.9

gets too big, e.g. 0.1 , does SSC have poorer performance. In this case, SSC prevents swapping of subtrees with similar but unequal semantics. We recorded how many subtree checks found a nonzero SSD smaller than 0.1; this happened approximately 20% of the time, misleading SSC. In general, we can see that LBSS is only required to be sufficiently small, and perhaps any value under 10^{-2} would be suitable.

The third parameter investigated is the number of unsuccessful trials permitted in selecting semantically similar subtrees (MT). Values of MT from 8 to 20 keep the performance of SSC roughly consistent. When MT is too small, e.g. MT = 4, the performance of SSC is worse. This can also be understood by observing the percentage of SSC events that successfully exchanged two semantically similar subtrees. For MT=4, only 30% to 40% of SSC events successfully exchanged subtrees, while this figure rises to about 90% for MT=20. Thus further increasing MT may not help, because nearly all crossover events have already successfully exchanged semantically similar subtrees.

Overall, these results highlight some important issues in determining the values for SSC parameters. It seems that UBSS should lie in the range 0.2 to 0.8, LBSS should be less than 10^{-2}, MT in the range 8 to 20.

Tab. 5.13: Mean best fitness of SSM with different parameters. Note that the values are scaled by 10^2.

Mutations	F1	F2	F3	F4	F5	F6	F7	F8
SM	2.14	2.50	0.95	1.60	0.96	1.56	2.99	2.07
SSMU01	1.75	2.11	0.61	1.31	0.82	1.42	1.86	1.70
SSMU02	1.37	1.73	0.45	1.07	0.74	1.20	1.30	1.39
SSMU04	1.13	1.36	0.42	1.16	0.72	1.16	1.03	1.01
SSMU06	1.99	2.40	0.82	1.49	0.94	1.52	2.75	2.02
SSMU08	2.05	2.39	0.91	1.51	0.94	1.51	3.02	2.04
SSMU1	2.06	2.46	0.92	1.55	0.92	1.53	2.97	2.06
SSML1	1.43	1.74	0.58	1.14	0.85	1.44	1.36	1.21
SSML2	1.38	1.50	0.46	1.06	0.78	0.98	1.15	1.03
SSML3	1.13	1.36	0.42	1.16	0.72	1.16	1.03	1.01
SSML4	1.13	1.39	0.36	1.11	0.77	1.02	1.05	0.98
SSML5	1.10	1.49	0.37	0.97	0.76	1.17	1.06	0.98
SSMMT4	1.72	2.04	0.69	1.39	0.78	1.27	1.90	1.34
SSMMT8	1.36	1.78	0.48	1.03	0.77	1.15	1.20	1.09
SSMMT12	1.13	1.36	0.42	1.16	0.72	1.16	1.03	1.01
SSMMT16	1.07	1.38	0.33	0.95	0.67	1.13	1.16	0.89
SSMMT20	1.06	1.35	0.36	1.00	0.70	0.91	0.98	0.99
SSMMT24	0.98	1.33	0.35	0.88	0.78	1.13	1.01	0.90

5.5.2 Parameters Sensitivity Analysis of Semantic Similarity based Mutation

This subsection examines the impact of changing parameters to the performance of semantic similarity based mutation, SSM. The GP parameters were setup as in Section 5.3. Similar to SSC, three parameters of SSM were investigated, namely, the *lower bound of semantic sensitivity* (LBSS), the *upper bound of semantic sensitivity* (UBSS), and Max_Trial (MT) used to selection two semantically similar subtrees. Configuration and conventional name for SSM are similar to SSC.

We also recorded the best fitness of a run, averaged over 100 runs, and the results are shown in Table 5.13. The mean best fitness of standard mutation is also shown in the top row of this table.

Similar to SSC, UBSS has a remarkable effect on the performance of SSM. It seems

Tab. 5.14: The percentage of SSM that successfully exchange two semantically similar subtrees.

Mutations	F1	F2	F3	F4	F5	F6	F7	F8
SSMU01	15.9	15.6	16.5	15.2	14.6	14.3	17.3	16.8
SSMU02	40.4	39.9	41.0	40.3	38.9	39.2	35.2	34.5
SSMU04	68.4	68.1	68.7	68.2	66.9	70.0	91.3	90.9
SSMU06	100	100	100	100	100	100	100	100
SSMU08	100	100	100	100	100	100	100	100
SSMU1	100	100	100	100	100	100	100	100
SSMMT4	30.6	30.5	30.8	30.5	29.7	29.9	55.4	55.1
SSMMT8	52.9	52.8	53.2	52.7	51.8	51.9	80.4	80.3
SSMMT12	68.4	68.1	68.7	68.2	66.9	66.9	91.3	90.9
SSMMT16	78.8	78.5	79.1	78.7	77.3	77.7	96.0	95.8
SSMMT20	85.4	85.5	85.8	85.5	84.8	84.7	98.1	98.2
SSMMT24	90.3	90.1	90.3	89.9	89.5	89.6	99.1	99.0

that values from 0.2 to 0.4 are suitable for the problems under test, with values of 0.4 being the best. It is different with SSC where the suitable range of UBSS is lager (from 0.2 to 0.8) and the best range could be from 0.2 to 0.6. If UBSS of SSM is too small (0.1) or too big (0.8, 1) the performance of SSM is poorer. This can also be explained by recording the percentage of SSM that successfully selects two semantically similar subtrees, as shown in Table 5.14. We can see that if UBSS is too small, only a few SSCs can succeed in exchanging semantically similar subtrees (from 30% to 40% when UBSS is 0.1), so that SSM underperforms (trying to increase the Max_Trial to compensate for decreasing the upper bound is also unsuccessful). By contrast, if UBSS is too large, it is almost trivial to find semantically similar subtrees (almost 100% for UBSS=0.6, 0.8, 1) because most subtrees are sufficiently semantically similar, so that SSM behaves like SM.

Likewise, LBSS has almost no effect on the performance of SSM provided it is sufficiently small. Table 5.13 shows that while LBSS was set to small values (from 10^{-2} to 10^{-5}), the performance of SSM was almost unchanged. The reason is similar to SSC. Only in the case when LBSS gets too big, e.g. 0.1 , does SSM have poorer performance. In this case, SSM prevents swapping of subtrees with similar but unequal semantics, this happened

approximately 20% of the time, misleading SSM.

With the third parameter, the number of unsuccessful trials permitted in selecting semantically similar subtrees (MT), the values from 8 to 24 keep the performance of SSM roughly consistent, in that the values of MT from 12 to 24 seem better than the left values . When MT is too small, e.g. MT = 4, the performance of SSM is worse and be understood by observing the percentage of SSM events that successfully exchanged two semantically similar subtrees. For MT=4, only 30% to 40% of SSM events successfully exchanged subtrees, while this figure rises to about 90% for MT=24. Thus further increasing MT may not help, because nearly all mutation events have already successfully exchanged semantically similar subtrees.

Overall, for SSM, it seems like, UBSS should lie in the range 0.2 to 0.4, LBSS should be less than 10^{-2}, MT in the range 12 to 24.

5.6 Conclusions

This chapter presented an investigation of semantic diversity and semantic locality of GP operators. The experimental results showed that while promoting semantic diversity of operators (crossover and mutation) has a limited effect on the performance of GP, improving semantic locality often leads to a significant improvement of GP performance. The results also demonstrated that when using a high rate of crossover and mutation, simultaneously improving semantic locality of both crossover and mutation helps to further enhance the performance of GP compared to purely doing for only crossover or mutation.

In the last section, the effect of some parameters on the performance of SSC and SSM were analysed. The experimental results showed that a wide range of these values could be suitable for these operators with some best values indicated for each operator on the problems examined.

Finally, we finish this chapter with some notes. Since, crossover has long been seen as the primary operator of GP [98], in the following chapters, we only focus on studying semantic based crossovers. For this reason, crossover rate is always set at 0.9 and a small

rate of mutation at 0.05 is set with the aim to study crossover in the normal form of GP. The next chapter will propose some approaches to improve Semantic Similarity based Crossover (SSC).

Chapter 6

Improving Semantic Similarity based Crossover

The previous chapter showed the important role of semantic locality for crossover and mutation on GP performance. This chapter presents some methods to improve Semantic Similarity based Crossover (SSC). First, another way to collect sample semantic points, based on the fitness cases, is introduced. A representation that is based on attributes to store the above semantics is used. Next, two approaches to improve SSC are proposed. Then, the comparative results between the two new improved versions of SSC with with SSC are presented. Finally, a method for reducing the number of attributes needed to store semantics is proposed. The results in this chapter have been published in [134, 138].

6.1 Measuring Semantics

This section presents an alternative way to sample semantics of a subtree and the use of attributes to store this semantics.

6.1.1 Subtree Semantics

Although, the use of random sampling for measuring subtree semantics in Chapter 4 has a number of benefits to GP, it has a drawback: It takes time to quantify and compare the semantics of two subtrees during the crossover operation. This is exhibited in that the average time of a run with SSC is often higher than one with SC. To overcome this, in this chapter, we use a simpler way for calculating the subtree semantics that is based on the fitness cases of the problem. This semantics is called *Subtree Semantics*. Subtree semantics used in this chapter is similar to McPhee's semantics [123] but extended to real valued problems rather than Boolean problems. Formally, the subtree semantics of any (sub)tree is defined as follows:

Definition 2 Let $P = \{p_1, p_2, ...p_N\}$ be the fitness cases of the problem on domain D and let F be a function expressed by a (sub)tree T on D. Then the *Subtree Semantics* of T on domain D is the set $S = \{s_1, s_2, ..., s_N\}$ where $s_i = F(p_i), i = 1, 2, .., N$. □

This way of measuring semantics is a special case of sampling semantics in Chapter 4. The difference between subtree semantics and sampling semantics is that sampling semantics is measured based on a number of random points on the problem domain while subtree semantics is always evaluated based on the fitness cases. It is also noted that, although subtree semantics is calculated based on fitness case points in the same way as Krawiec and Lichocki's semantics [103], our work is different from Krawiec and Lichocki's work in several ways. First, Krawiec and Lichocki only use the semantics of the whole trees while in this thesis, the semantics is at subtree level. Second, we guide crossover by considering the semantic relationships between subtrees rather than the semantic relationship between offsprings and their parents. Last, there are a number of children generated in Krawiec and Lichocki's work, but only one is added to the next generation, while in our crossover there are always two children being generated and transfered to the next generation. Therefore, our crossover is less time consuming than the crossover of Krawiec and Lichocki [103].

Based on subtree semantics, a semantic distance between two subtrees is defined in a similar way to sampling semantics distance (SSD) in Chapter 4. From that, two semantic

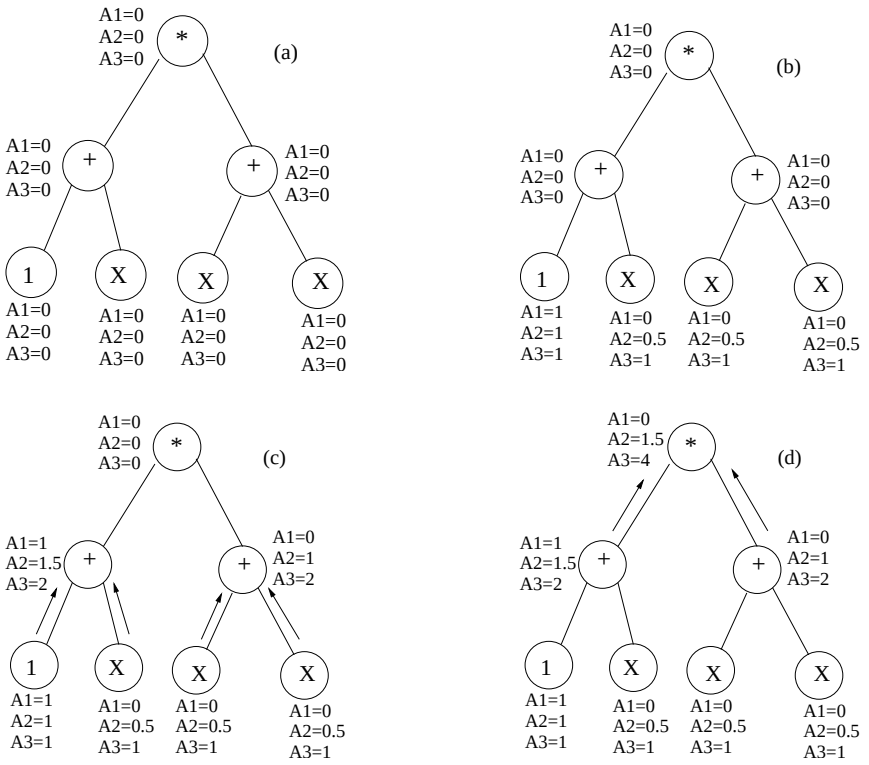

Fig. 6.1: An individual in AGP and the process of evaluating the value of its attributes. This illustrates the initialisation (a) and the evaluation of its attributes (b, c, d) by propagation up toward the rote note.

relationships between subtrees, semantic equivalence (SE) and semantic similarity (SSi) are defined in the same way with their definitions in Chapter 4.

6.1.2 Attributes-based Representation

As mentioned in the previous section, the subtree semantics of each subtree is stored in the root node of that subtree by using a number of attributes and the resulting GP system is called *Attributes Genetic Programming* (AGP). Assume that the problem has N fitness cases, then N attributes are added to every node in a tree-based individual. In Figure 6.1 N is set to 3, so three attributes A_1, A_2, A_3 are added to every node in the individual to store semantics of the subtree rooted at that node.

Figure 6.1 also describes the process of evaluating attribute values in AGP. Initially (Figure 6.1a), the attributes are set to zero. Assume that the fitness cases include three

values 0, 0.5, and 1, then, in the second step, the attributes of the leaves of the individual are assigned with these values (Figure 6.1b, attributes at the nodes labeled with a constant are assigned with the value of that constant). Next, the attributes at the level above the leaves are assigned with values. At this point, the semantics of the leaves is passed upward to their parents, and the operator at those nodes are applied to calculate the values for the attributes (Figire 6.1c) at these nodes. This process is then continued until the attributes at the root node are assigned with values (Figure 6.1d). It is noted that when this process of value propagation completes, the fitness of the individual can be obtained by comparing the semantics of the root node with the values of the target function on the corresponding fitness cases. This helps to speed up crossover in AGP in comparision with using sampling semantics.

The power that subtree semantics brings to AGP comes with a cost that is the increase of storage size of the population. Assume that the population is of P individuals. If the average size of an individual in the propulation is S (nodes) and there are N fitness cases (N attributes) which occupies B bytes each, then, the magnitude of increase in storage size, (I), in AGP is calculated as in Equation 6.1. For instance, if each attributes is stored in a real number with 4 bytes, then the memory space needed to store subtree semantics of all individuals in GP population, with typical population sizes, is shown in Table 6.1. It is generally well within the capabilities of modern computers.

$$I = P * S * N * B \tag{6.1}$$

6.2 Improving Semantic Similarity based Crossover

Although, the previous chapter has shown some advantages of SSC over other crossover operators, it comes with a cost, especially when being compared with SC, it has extra parameters to tune. The most important extra tunable parameter in SSC are the semantic sensitivity bounds. As the performance of SSC depends on the semantic sensitivities,

Tab. 6.1: Memory space needed to store Subtree Semantics.

Pop_Size	Fitness_cases	Average_Size	Memory_Space (Mbs)
500	20	100	3.18
	100	100	19.1
	500	100	95.4
1000	20	100	7.63
	100	100	38.2
	500	100	190

incorrect selection of these values could affect SSC performance. In this section we propose two methods for remedying this drawback. The first approach allows these sensitivities to be self-adapted during the evolutionary process and the second approach eliminates the upper bound of sensitivity from SSC. It should be noted that in the two following subsections, we mostly focus on controlling the upper bound of semantic sensitivity of SSC. This parameter is more important and more difficult to determine than the lower one. The lower bound of semantic sensitivity is simply adjusted based on the upper one or set as a small non-zero value.

6.2.1 Self-Adapting Semantic Sensitivities

The first method to overcome the weakness of SSC allows semantic sensitivities of SSC to be self-adapted. There have been a number of studies of automated control of different aspects of evolutionary algorithms. Angeline [4] distinguished three categories:

1. *component-level* – adaptive approaches that automatically alter some elements of an individual [5].

2. *individual-level* – adaptive techniques that adjust the way an individual itself is handled by the system [144, 63].

3. *population-level* – a class of methods that gathers the global information from the whole population and use this information to dynamically adapt parameters [162, 168].

6.2. IMPROVING SEMANTIC SIMILARITY BASED CROSSOVER

Most early research on self-adaptation focused on controlling operators (crossover and mutation). More recently, self-adaptive schemas have also been developed for other aspects. One parameter that has been attracted much attention is the population size [40, 73]. Silva and Dignum [175] presented a self-adaptive method for setting the maximum length of GP individuals.

The self-adaptive methods proposed in this subsection can be seen as population-level methods, in that they gather information from the whole population and use this information in the next generation. They differ from previous self-adaptive schemas in self-adapting two new parameters, the *semantic sensitivities*, in SSC.

The motivation for this is to guarantee that there is a certain portion of SSC operations that successfully exchange two semantically similar subtrees. It is easy to see that if the upper bound of semantic sensitivity is two small (e.g 10^{-2}), SSC may not work. The reason is that it is very difficult to select subtree pairs that satisfy SSC's condition for being swapped. Conversely, if the upper sensitivity is two great (e.g 100) SSC would behave almost as the same SC, as all most all subtree pairs could satisfy this condition. For this reason, a self-adaptive scheme called *Self-Adaptive Successful Execution* (SASE) is introduced. The main aim of SASE is to guarantee that there is a large enough portion, but not all, of SSC operations that successfully exchange similar subtrees pairs. Formally, the upper bound sensitivity is controlled according to the following equation:

$$\beta^{t+1} = \begin{cases} c_i \cdot \beta^t & \text{, if } {p_s}^t < \psi; \\ c_d \cdot \beta^t & \text{, if } {p_s}^t > \psi; \\ \beta^t & \text{, if } {p_s}^t = \psi. \end{cases} \tag{6.2}$$

where β^t is the upper bound of semantic sensitivity at generation t, β^{t+1} is the value at generation $t+1$, and ${p_s}^t$ is the ratio of SSC operations that successfully exchange two similar subtrees at generation t. c_i is the size of increment and c_d is the size of decrement.

After the upper bound of semantic sensitivity gets changed, the lower bound of semantic sensitivity at generation $t+1$ is recalculated as: α^{t+1}=10^{-3}.β^{t+1}. The initial value of upper bound of sensitivity was set at 0.4 in the experiments in this chapter. Several values of ψ will be investigated in the following subsections.

6.2.2 The Most Semantic Similarity based Crossover

While the first approach to improve SSC tries to self-adapt SSC's semantic sensitivities, the second method eliminates its semantic sensitivities. This results in a new semantic based crossover. The new crossover is called the Most Semantic Similarity-based Crossover (MSSC). The idea behind MSSC is to avoid the manual tunning of the semantic sensitivities in SSC, while still maintaining a semantically small change. MSSC works as follows. N subtree pairs are randomly selected from the two parents. The subtree semantic distance (SSD) of the subtrees in each pair is then calculated. The pairs that have the smallest SSD (but not equivalent) in the N pairs is chosen for crossover. In MSSC, the concept of semantic equivalence is the same as in SAC. Algorithm 5 shows how MSSC works in detail. In the experiments of MSSC, Extremal_Value was set to 10^6 and several values of Max_Trial (TM) was tested.

6.3 Experimental Settings

To examine the effects of the improved versions of semantic similarity based crossover, SSC, in comparison with other crossover operators, we tested them on the eight real-valued symbolic regression problems in Chapter 5. The basic experimental settings is as in Table 5.2.

In this chapter, the performance of two new improvements of SSC will be compared not only with standard crossover, semantic similarity based crossover, but also with some methods that are based on fitness including No Same Mate (NSM) selection [62] and Soft Brood Selection (SBS) [3].

Algorithm 5: The Most Semantic Similarity based Crossover

```
select Parent 1 P_1;
select Parent 2 P_2;
Count=0;
Max=Extremal_Value;
while Count<Max_Trial do
    choose a random crossover point Subtree_1 in P_1;
    choose a random crossover point Subtree_2 in P_2;
    SD=SSD(Subtree_1, Subtree_2)
    if α < SD < Max then
        Max=SD;
        CrossPoint1=Subtree_1;
        CrossPoint2=Subtree_2;
Execute crossover by exchange the subtrees at CrossPoint1 and CrossPoint2;
```

For SSC, the lower bound of semantic sensitivity is fixed at 10^{-3}. It has been shown in Chapter 5 that the performance of SSC is mostly consistent with any small enough value of the lower bound of semantic sensitivity. The upper bound of semantic sensitivity is fixed at 0.4. This value has been shown as one of the best value for the good performance of SSC. Three values of Max_Trial of SSC, 12, 16, 20 are tested. These values were calibrated from the experiments in the previous chapter as some of the best values. Again, three experiment configurations of SSC will be called SSCX with X = 12, 16, 20 respectively.

For SBS, three brood sizes: 2, 3, and 4 are used. Although, Tackett suggested the use of a subset of fitness cases to figure out the individual in the brood to the next generation [180, 179], in this comparison we use the original version of SBS of Altenberg in [3]. In this, the number of the fitness evaluation of SBS increases *pCross*N* times (where, N is brood size and *pCross* is the rate of crossover) in comparison with the standard GP crossover. Therefore, for a fair comparison, the population size of the GP with SBS should be *pCross*N* times reduced. In this paper, three population sizes: 277, 185, 138 corresponding to these above brood sizes are used. This way of comparing a crossover with SBS is similar to what have been used in some previous studies [116, 103]. Three configurations of SBS will be referred as SBSX with X= 2, 3, 4.

Tab. 6.2: The comparison of two improved schemas with SSC, SBS, NSM and SC in terms of number of successful runs out of 100 runs.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	12	4	40	3	20	17	31	18
NSM	19	7	50	8	21	20	34	11
SBS2	28	14	60	10	42	19	32	19
SBS3	24	10	58	17	32	23	32	20
SBS4	17	7	39	17	38	22	25	19
SSC12	30	17	70	23	49	38	43	32
SSC16	31	12	72	21	50	39	55	20
SSC20	27	14	71	17	52	41	39	25
SASE65	31	17	76	29	61	42	75	59
SASE75	32	16	85	26	60	38	75	49
SASE85	36	15	81	33	58	37	71	53
SASES	37	16	77	41	61	39	83	61
MSSC12	35	19	79	34	55	45	79	40
MSSC16	29	14	80	33	62	41	70	45
MSSC20	40	18	88	39	72	44	65	37

For SASE, the size of increment (c_i) was fixed at 0.9 and the size of decrement was set as $1/c_i$. These values were calibrated from our experiments as good values for SASE. Four configurations of SASE were tested. In the first three configurations, the ratio of SSC that successfully exchange (referred to as successful SSC) two semantically similar subtrees of 65%, 75%, and 85% were examined. These configurations of SASE are called SASEX with X as 65, 75, and 85, respectively. In the last version of SASE, the rate of successful SSC is adapted itself. The motivation for this is that at the beginning of the search process, GP tends to need more exploratory power to discover new solution areas. Therefore, the rate of successful SSC in earlier generations should be smaller. Later on in a run, SSC should be encouraged to more exploitative. In later generations, the rate of successful SSC should be increased. In order to implement this, the rate of successful SSC will be changed according to the following equation:

$$R\_SSC = \alpha + (\beta - \alpha) \cdot \frac{Current\_Gen}{Max\_Gen} \tag{6.3}$$

where $Current\_Gen$ is the current generation and Max_Gen is the maximal number of generations set for the algorithm. R_SSC is the rate of successful SSC in the current generation. In our experiments, α was set at 65% and β was set at 85%. This version of SASE is referred to as SASES.

For MSSC, the lower bound semantic sensitivity was set as in SSC at 10^{-3}. Three values of Max_Trial: 12, 16, 20 were tested resulting in three configurations of MSSC referred as MSSCX with X=12, 16, 20.

6.4 Results and Discussion

6.4.1 Results

For all systems, two classic performance metrics, namely mean best fitness and the number of successful runs were recorded. The results of the number of successful runs (out of 100 runs) of these crossovers are presented in Table 6.2. Table 6.3 shows the best fitness found, averaged over all 100 runs of each GP system. We tested the statistical significance of the results in Table 6.3 using a Wilcoxon signed-rank test with a confidence level of 95%. The results in Table 6.3 are printed as follows:

1. If a run of NSM, SBS, SSC, SASE, and MSSC is not significantly better than SC, it is printed in plain face

2. If a run of NSM, SBS, SSC, SASE, and MSSC is significantly better than SC, its is printed *italic* face

3. If a run of SASE and MSSC is significantly better than all SC, NSM, SBS, and SSC it is printed ***bold and italic*** face.

Tab. 6.3: The comparison of two improved schemas with SSC, SBS, NSM and SC in terms of mean best fitness. Note that the values are scaled by 10^2.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.54	2.02	0.60	1.43	0.76	1.18	1.68	1.21
NSM	1.28	1.58	0.52	1.21	0.76	1.04	1.42	1.24
SBS2	*1.00*	*1.48*	*0.39*	*0.94*	*0.57*	1.00	*1.31*	0.93
SBS3	*1.18*	*1.55*	*0.46*	1.11	0.61	1.09	1.45	1.38
SBS4	1.40	1.67	0.69	*0.93*	0.60	1.26	1.91	1.22
SSC12	*0.82*	*1.08*	*0.32*	*0.80*	*0.42*	*0.65*	*0.97*	*0.95*
SSC16	*0.80*	*1.14*	*0.31*	*0.81*	*0.43*	*0.63*	*0.75*	*0.89*
SSC20	*0.86*	*1.09*	*0.32*	*0.82*	*0.45*	*0.55*	*0.92*	*0.79*
SASE65	*0.74*	*1.03*	*0.20*	*0.75*	*0.35*	*0.49*	*0.61*	***0.55***
SASE75	*0.71*	*1.02*	***0.17***	*0.76*	*0.37*	*0.53*	***0.56***	***0.50***
SASE85	*0.74*	*1.02*	*0.22*	*0.66*	*0.38*	*0.51*	***0.47***	***0.45***
SASES	*0.70*	*0.89*	*0.21*	*0.63*	***0.32***	*0.52*	***0.30***	***0.43***
MSSC12	*0.71*	***0.85***	*0.20*	*0.60*	*0.37*	*0.52*	***0.29***	***0.54***
MSSC16	*0.77*	*0.87*	***0.19***	*0.62*	*0.35*	*0.49*	***0.30***	***0.48***
MSSC20	*0.70*	*0.92*	***0.17***	*0.61*	***0.30***	*0.50*	***0.47***	***0.59***

6.4.2 Discussion

Table 6.2 and Table 6.3 show some important results. Firstly they are consistent with the results in Chapter 5 where promoting semantic diversity like NSM gives a positive effect on the performance of GP. It can be seen from Table 6.2 that NSM often finds solution easier than SC. The number of successful runs of NSM is usually higher than those of SC with only one exception on function F_8. Table 6.3 shows that the quality of solutions that found by NSM is often better than ones found by SC. However, the table also show that the size of the improvement is not large and not statistically significant.

The results in the two above tables are also consistent with the results in Chapter 5 when we compare the performance of SSC with SC. It can be observed from Table 6.2 that the number of successful runs found by SSC is always much higher than those found by SC. Table 6.3 shows that SSC is always significant better than SC in terms of finding better solutions. Generally, these tables confirm the important role of improving semantic locality of crossovers as SSC even the way for measuring semantics of subtrees has changed.

Tab. 6.4: Average time of a run in milliseconds of MSSC, SASE, SSC, SBS, NSM and SC.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	3042	3092	3112	4022	3428	3646	10180	10128
NSM	2991	3054	3050	4029	3213	3657	10786	10164
SBS2	6141	6387	4952	6913	5477	6425	19924	21250
SBS3	5838	5849	5046	6176	5104	5418	14668	16500
SBS4	4580	4348	3641	4830	3950	4106	13044	14906
SSC12	3117	2927	3107	4016	3537	3696	10343	10572
SSC16	3126	2953	3098	3972	3418	3623	10067	10037
SSC20	3215	3229	3373	4116	3658	3833	10027	10474
SASE65	2958	3244	3240	4103	3682	3935	6269	7883
SASE75	2837	2975	3020	3978	3309	3629	7922	8448
SASE85	2917	3011	2622	3707	3153	3438	9019	9376
SASES	2867	2879	2773	3726	3266	3439	7154	8077
MSSC12	2979	3006	3056	3991	3367	3425	6480	8399
MSSC16	2954	2901	3088	3825	3240	3558	5643	7149
MSSC20	2672	2673	2553	3787	3224	3134	5560	6202

For SBS, it can be seen from the table that SBS helps to improve the performance of GP in comparison to SC. The table shows that the number of solutions found by SBS is often greater for SC and NSM. It is, however, very important to point out that improving semantic locality in SSC is still better than searching based on fitness as in SBS. The number of the solutions found by SSC is consistently greater than ones found by SBS.

Table 6.3 again shows that SBS is better than SC in terms of finding solutions with higher quality (smaller of the best fitness). However, the improvement of SBS is not so large and not always statistically significant. The Wilcoxon signed-rank test results show that SBS is only significantly better than SC with some brood sizes and on some functions like F_1, F_2, F_3, F_4, F_5 and F_7. On the two other functions, F_6, F_8, no improvement of SBS over SC is observed. By contrast, SSC is always significantly better than SC on all problems with all tested values of Max_Trial.

For the two new improved methods, it can be seen from Table 6.2 that both SASE and MSSC help to further improve GP performance. This is reflected by the greater number of solutions found by SASE and MSSC over SSC. In four configurations of SASE it seems

the performance of SASES was the most consistent while the performance of three MSSC's configuration was mostly the same.

The results in Table 6.3 are strongly consistent with those in Table 6.2 confirming the superiority of SASE and MSSC over SSC. The improvement of SASE and MSSC over SSC seem more impressive on the two bivariate functions, F_7 and F_8. The Wilcoxon signed-rank test results confirm that the improvement of all SSC based crossovers is statistically significantly better than SC and that in many cases, the improvement of SASE and MSSC over SSC was also significant. The results in the table also show that MSSC was slightly better than SASE.

All in all, the results in this section show that improving semantic locality as in SSC is better than two methods that are based on fitness control NSM and SBS in terms of enhancing GP performance. Moreover, two new improvements of SSC, SASE and MSSC, help not only to overcome the weakness of SSC (by removing the manual tunning of extra parameters) but also to further improve SSC performance.

Since, subtree semantics is stored in the attributes, it is important to see how effective it is in reducing semantic checking time. We measured the average time of a run of the above methods. The results are show in Table 6.4. It can be seen from this table that the average running time of NSM is mostly equal to the average running time of SC. It is understandable since NSM does not need any extra time for fitness qualifying. By contrast, the average running time of SBS is often much higher than of SC, especially with the brood size of 2 and 3. The reason could be that SBS need more time to evaluate the context of subtrees before doing crossovers.

For SSC, the table shows that its average running time is almost the same with that of SC. It can be seen from this table that sometimes SSC runs faster than SC and sometimes SC runs faster than SSC. These results are more positive than the results in Chapter 5 where it has been shown that SSC always runs more slowly than SC. This confirms the benefit of using attributes to store trace semantics.

With two new improved methods of SSC, the table shows that SASE and MSSC usually run faster than SC. The exception only lies in one case of SASE where SASE65 runs more

slowly than SC. Generally, the average running time of SASE and MSSC was only equal to 80% the average running time of SC. In the group of bi-variable functions, the running time, compared to SC, was often reduced nearly a half with MSSC. In a comparison among SASE and MSSC, it can be seen that MSSC usually runs faster than SASE. It can be explained by the results in the next chapter where MSSC is shown to produce less bloat than SASE and other crossovers.

6.5 Attribute Reduction

Since AGP used a number of attributes added to every node in every individual to store semantic, it consumes more memory space than standard GP. The experiments in this section are designed to show a way to overcome the above limitation. In particular, an approach that uses only a subset of fitness cases as subtree semantics is proposed. In other words, the subtree semantics of a subtree (tree) is now redefined as the following subsection:

6.5.1 Subtree Semantics

Definition 3 Let $P = \{p_1, p_2, ...p_N\}$ be the fitness cases of a problem on domain D, then $Q = \{q_1, q_2, ...q_K\}$ is a randomly drawn subset of P. Let F be the function expressed by a (sub)tree T on D. Then the *Subtree Semantics* of T on domain D is the set $S = \{s_1, s_2, ..., s_K\}$ where $s_i = F(q_i), i = 1, 2, .., K$. □

With this definition of subtree semantics, the semantic distance, semantic relationships and semantics-based crossovers are redefined accordingly. The only modification is that the number of attributes needed to store subtree semantics is now reduced from N to K with $K \leq N$. Assume that the population is of P individuals. If the average size of an individual in the population is S (nodes) and there are N fitness cases which occupies B bytes each, then, the magnitude of reducing in storage size, (R), in AGP is calculated as in Equation 6.4

$$I = P * S * (N - K) * B \tag{6.4}$$

Notice that the values for these attributes can be calculated during the fitness calculation in a similar way as in Subsection 6.1.2. The following experiments and results will show to what extent one can reduce the number of the attributes for AGP on the tested problems so as to reduce the memory space needed to store semantics while maintaining the performance of AGP with semantics based crossover operators.

6.5.2 Experimental Settings

As has been shown in Table 6.1, the memory space needed to store subtree semantics is only really significant when the number of fitness cases is large. Therefore, in this experiment, we increased the number of fitness cases of the problems. In particular, the number of fitness cases for single variable problems was set at 80 and for bivariate problems was 400. We tested the performance of SSC, SASE, and MSSC with this new and reduced semantics and compared them with SC.

For SSC, we selected the first configuration where the upper bound of semantic sensitivity was set at 0.4. The proportion of fitness cases chosen for subtree semantics was set as Number_of_Fitness_Cases/X where X = 1, 2, 4, or 8. These four configurations of SSC will be called: SSCX with X=1, 2, 4, 8. For SASE, SASES was selected to test and for MSSC, MT was set as 12. Similar to SSC, four subsets of fitness cases were examined and will be refered as SASEX and MSSCX with X=1, 2, 4, and 8, respectively.

6.5.3 Results and Discussion

To compare the performance of these configurations with SC, we recorded the best fitness of each run and averaged over 100 runs. The results are shown in Table 6.5.

Again, a Wilcoxon signed-rank test with a confidence level of 95% is conducted in Ta-

Tab. 6.5: The comparison between SC and crossovers for improving semantic locality with semantics is defined on a subset of fitness cases in terms of mean best fitness. Note that the values are scaled by 10^2.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.47	2.00	0.64	1.53	0.91	1.65	1.42	1.19
SSC1	0.97	1.09	0.28	0.89	0.48	0.69	1.25	0.77
SSC2	0.98	1.24	0.29	0.94	0.45	0.70	1.03	0.78
SSC4	0.92	1.16	0.27	0.88	0.48	0.71	1.03	0.79
SSC8	0.90	1.24	0.30	0.90	0.52	0.72	1.10	0.78
SASE1	0.92	1.08	0.22	0.71	0.38	0.65	0.55	0.69
SASE2	0.89	1.09	0.18	0.74	0.42	0.64	0.49	0.60
SASE4	0.87	1.09	0.22	0.79	0.41	0.66	0.66	0.66
SASE8	0.85	1.06	0.23	0.82	0.37	0.62	0.45	0.65
MSSC1	0.81	0.99	0.16	0.70	0.37	0.63	0.35	0.56
MSSC2	0.78	1.07	0.19	0.66	0.41	0.61	0.34	0.61
MSSC4	0.69	1.06	0.16	0.68	0.42	0.60	0.42	0.53
MSSC8	0.79	0.99	0.22	0.84	0.36	0.62	0.46	0.63

ble 6.5, and if a crossover configuration was not significantly better than SC, it is printed in *italic* face. It can be seen from the table that all three semantic locality promoting crossovers were still significantly better than SC. Moreover, even when the number of samples in subtree semantics was reduced to 1/8 of the number of fitness cases, the performance of these crossover are still almost completely preserved. Therefore, we argue that using attributes to store semantics is a reasonable method. It not only facilitates the design of semantic based crossovers and guarantees the execution of these crossovers, but also helps to speed up the GP system. Moreover, whenever, the number of fitness cases is too large, we can use a much smaller subset of them as semantics without degrading the performance of semantic based crossovers. This table also confirms the superior performance of SASE and MSSC over SSC.

6.6 Conclusions

This chapter introduces an alternative way to sample points for measuring semantics of a subtree that is based on the fitness cases of the problems. This way of sampling semantics allows us to use a number of attributes to store semantics of a subtree. By caching semantics, it helps to speed up semantic relationships checking and facilitates the design of some new semantic based crossovers.

After that, two new improvements of SSC are proposed. The first one self adapts the semantic sensitivities of SSC and the second version tries to eliminate the higher sensitivity from SSC. The experimental results showed that these two approaches helped not only to remove the burden of manually tunning semantic sensitivities in SSC, but also to further improve SSC performance.

Last, we proposed a way for reducing the memory space needed to store semantics. Here, subtree semantics is measured on a subset that is randomly drawn from the fitness cases of the problems. An experiment was dedicated to investigate the efficiency of this method. The results showed that we could use a much smaller set of the fitness cases while still preserving the performance of semantic locality promoting crossovers. The next chapter will present an investigation of some properties (semantic diversity, semantic locality etc.) of semantic based crossovers to understand the reason for the advantages of these operators.

Chapter 7

Some Properties of Semantic based Crossovers

The previous chapters showed that promoting semantic diversity and locality of crossover brings positive impact on GP performance. This chapter analyses some characteristics of semantic based crossovers. They include the rate at which semantically equivalent crossover events occur, the semantic diversity resulting from such crossovers, the locality of the operator, its constructive effect, its code bloat effect, and semantics exchanged in crossovers. The results are compared with Standard Crossover (SC) and No Same Mate (NSM) selection [1]. The results in this chapter shed some light into the performance of semantic based crossovers. In particular, it helps to explain why promoting semantic diversity can only slightly improve GP performance while enhancing semantic locality leads to a significant improvement. The results in this chapter have been published in [133, 140].

The GP parameter settings in this chapter are described in Table 5.2 in Chapter 5. Three configurations of Semantic Aware Crossover (SAC) were used, with semantic sensitivities set to 10^{-X} with X=2, 3, and 4. These configurations of SAC are denoted as SACX, for X=2, 3, or 4. For Semantic Similarity based Crossover (SSC), LBSS was set to 10^{-3} and UBSS to 0.4. Three configurations of SSC were used, with Max_Trial being

[1] Soft Brood Selection (SBS) is not considered in here as it is a local search based technique making it irrelevant for the comparison in this chapter.

at 12, 16, and 20. For Self-Adaptive Successful Execution (SASE) method and the Most Semantic Similarity based Crossover (MSSC), the experimental settings are the same with those in Chapter 6.

7.1 Rates of Semantically Equivalent Crossover Events

The first experimental result records the extent of semantically equivalent exchanges arising from the tested crossover operators. Here we say that a crossover operation is an equivalent crossover if it is performed by exchanging two semantically equivalent subtrees. Since the proposed new crossover operators (SAC and SSC, SASE, and MSSC) work by analysing the semantics of subtrees, and trying to prevent the exchange of semantically equivalent subtrees, it would be informative to see how frequently this actually happens. This information shows us how frequently SC fails to change the semantics of individuals (semantically unproductive) and the extent to which SAC, SSC, SASE, and MSSC could solve this problem. The statistics collected are the percentage of such crossover events, for all crossovers. The results are shown in Table 11.3.

It can be seen from Table 11.3 that the overall average for equivalent crossovers in SC is from 12% to 17%. This value for NSM is only slightly smaller ranging from 11% to 15%. By contrast, it is much smaller (from 1% to 2%) for SAC and from 1% to 6% for SSC, SASE and MSSC. It is clear that SAC, SSC, SASE, and MSSC are more semantically exploratory than SC on these problems. We also conducted an experiment to test how crossover affects the relative fitness of the offspring compared to their parent when it swaps two semantically equivalent subtrees. The results indicate that, in nearly all cases (about 99%), such crossovers leave the child fitness unchanged. It should be noted that swapping two semantic equivalent subtrees does not always leave the fitness of the children unchanged, because two semantically equivalent subtrees are not necessarily identical subtrees. If they are only approximately the same, the crossovers can still change the fitness of the children, but the level of change is very small, often less then 1%.

We note also that the number of equivalent crossovers is often marginally higher for

Tab. 7.1: Average percentage of semantically equivalent subtrees in crossover

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	14.3	14.6	17.3	13.2	14.1	14.4	11.8	12.2
NSM	14.0	14.3	15.0	12.8	13.7	14.0	11.5	11.7
SAC2	1.66	1.62	2.68	1.41	1.58	1.73	1.07	1.05
SAC3	1.67	1.57	2.72	1.43	1.56	1.70	1.07	1.02
SAC4	1.64	1.58	2.52	1.44	1.54	1.65	1.06	1.01
SSC12	3.18	3.36	3.10	1.79	2.57	4.75	0.29	0.42
SSC16	2.28	2.50	2.80	1.04	1.76	3.92	0.18	0.29
SSC20	1.71	1.70	1.87	0.73	1.34	3.20	0.14	0.23
SASE65	5.43	5.48	7.72	4.36	4.84	4.96	5.55	5.03
SASE75	4.07	4.01	4.98	3.22	3.63	3.66	4.16	3.75
SASE85	2.53	2.63	5.77	2.07	2.36	2.28	2.31	2.46
SASES	3.83	3.78	5.11	2.99	3.49	3.50	3.68	2.60
MSSC12	1.42	1.51	2.16	1.85	3.15	2.41	0.93	0.95
MSSC16	1.37	1.44	2.07	1.81	3.01	2.44	0.64	0.69
MSSC20	1.42	1.35	1.87	1.95	2.80	2.18	0.54	0.65

SSC and SASE than for SAC. This is understandable, as SAC focuses purely on preventing semantically equivalent crossovers, while SSC and SASE are also aimed at preserving semantic similarity of the exchanged subtrees. Preventing equivalent crossovers is valuable in reducing the unproductive crossovers of SC, but keeping semantic changes small (as seen in the succeeding sections) is even more effective in preserving diversity and achieving constructive effects, leading to substantial performance improvements.

7.2 Semantic Diversity

In the previous section, we have shown that SAC, SSC, SASE and MSSC encourage the exchange of semantically different subtrees, forcing a change in semantics relative to their parents. It triggers a question on how well do SAC, SSC, SASE and MSSC promote (semantic) diversity?

So far, two classes of metrics have been used to measure and control the diversity of a population: at the genotype level and at the phenotype level [61, 65]. The former is

Tab. 7.2: Average percentage of generating new children after applying SC, NSM, SAC and SSC, SASE and MSSC.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	65.0	66.3	61.6	58.8	64.8	58.8	68.3	69.7
NSM	67.5	67.4	64.6	62.2	65.3	62.4	69.1	70.9
SAC02	73.2	73.3	70.5	62.8	69.6	66.6	74.3	75.1
SAC03	72.7	75.3	69.3	64.5	69.8	67.5	73.0	74.7
SAC04	72.8	74.1	69.6	64.5	70.4	68.2	72.9	74.9
SSC12	75.7	77.9	70.9	70.7	69.8	67.9	73.7	76.9
SSC16	76.7	77.1	71.8	71.4	70.8	67.4	71.1	77.4
SSC20	77.0	78.0	70.7	72.4	70.4	67.3	73.6	77.1
SASE65	74.7	76.1	68.5	67.1	70.2	64.4	74.8	75.3
SASE75	75.6	76.9	69.2	67.6	69.4	66.9	75.0	75.1
SASE85	78.2	77.5	68.8	69.7	69.5	69.4	74.1	73.9
SASES	76.4	76.8	70.8	69.5	70.5	68.1	74.2	74.8
MSSC12	78.1	80.9	73.1	71.5	70.8	70.0	78.5	75.3
MSSC16	79.2	80.7	73.9	72.8	69.7	69.1	80.7	79.4
MSSC20	77.8	81.2	74.5	71.3	71.3	70.4	83.3	78.4

based on the syntax (i.e., structure) of an individual [148] and the latter on the behaviour (i.e., fitness) of an individual [126]. In this thesis, we propose a new measure for semantic diversity known as *Semantic Diversity of Crossover* (SDC). SDC is different from other metrics in that SDC does not aim to measure the difference between the individuals in the same population, but rather to measure the difference between the individuals of two successive populations. In other words, SDC is a measure of how much individuals change semantically through crossover. Here, the semantic difference between individuals before and after crossover is again determined based on the set of semantic points of the problems.

We used SDC to measure the semantic diversity of the above methods by counting the percentage of the crossover events that generated semantically different offspring from their parents (we will call them new children). These results are shown in Table 7.2.

It can be seen from Table 7.2 that there are about 60% to 70% of SC that can change semantics from parents to children. By preventing the swapping of two semantically equivalent subtress, both NSM and SAC increase the probability of crossovers that can gener-

ate semantically newly children. However, while NSM only marginally rises the rate of crossover that can change semantics (about 2% to 4%), SAC makes it to a greater extent (about 8% to 10%).

Comparing between SAC and SSC, it can be seen that SSC is often better than SAC in terms of generating semantically newly children. It is also interesting to see that, although SAC was better than SSC in terms of preventing equivalent crossovers, by keeping semantic changes small, SSC was nevertheless often better than SAC at producing a more semantically diverse population. With two improved version of SSC, SASE and MSSC, it can be seen from the table that while SASE is mostly equal with SSC, MSSC is generally better than both SSC and SASE in changing semantics of individuals in crossover. We note that SAC, SSC, SASE and MSSC in general do not guarantee to generate semantically new offspring, even though they try to swap two semantically different subtrees. We believe this arises from some fixed-semantic subtrees as in the Boolean domain [123].

7.3 Semantic Locality

The next set of results show the locality property of SSC, SASE and MSSC compared to SAC and SC. Generally, using a representation with high locality is important for efficient evolutionary search [69, 169]. To date, most current GP representations and operators only focus on controlling syntactic locality [69] and semantic locality is still very small. Since, the new crossover operators (SSC, SASE and MSSC) differ from other crossover operators in that it attempts to control the scale of change in terms of semantics rather than syntax, it is informative and useful to see how they achieve semantic locality.

To compare the locality of SSC, SASE, MSSC, with SAC, NSM and SC, an experiment was conducted where the fitness change of individuals before and after crossover was measured and recorded. For example, suppose two individuals (P_1 and P_2) having fitness of 10 and 15 are selected for crossover respectively, and after the crossover operation their children are C_1 and C_2, where C_1 roots at P_1 and C_2 roots at P_2 (meaning that C_1 and C_2 are generated by replacing a subtree of P_1 and P_2, respectively, with a new subtree) have fitness

Tab. 7.3: The average change of fitness after crossover for SC, NSM, SAC, and SSC.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.18	1.26	0.89	0.62	0.78	1.40	0.52	0.48
NSM	1.02	1.18	0.71	0.64	0.84	0.96	0.47	0.49
SAC02	1.07	1.08	0.81	0.68	0.76	0.92	0.51	0.56
SAC03	1.03	1.11	0.78	0.64	0.72	0.94	0.59	0.50
SAC04	1.05	1.19	0.80	0.67	0.75	0.96	0.58	0.50
SSC12	0.65	0.49	0.31	0.29	0.32	0.72	0.30	0.16
SSC16	0.42	0.40	0.25	0.25	0.29	0.68	0.21	0.16
SSC20	0.50	0.43	0.20	0.20	0.19	0.65	0.23	0.16
SASE65	0.76	0.58	0.42	0.31	0.44	0.72	0.17	0.16
SASE75	0.51	0.69	0.37	0.25	0.34	0.62	0.16	.017
SASE85	0.40	0.62	0.35	0.28	0.30	0.55	0.16	0.14
SASES	0.45	0.58	0.24	0.35	0.32	0.72	0.14	0.15
MSSC12	0.48	0.35	0.27	0.25	0.35	0.50	0.09	0.13
MSSC16	0.34	0.32	0.16	0.27	0.23	0.51	0.07	0.16
MSSC20	0.35	0.25	0.16	0.37	0.25	0.46	0.07	0.10

of 17 and 9. The change of fitness of these individuals is $(Abs(17-10)+Abs(9-15))/2 = 7.5$. This value was averaged over the whole population and over 100 runs. The average fitness change of individuals before and after crossover is shown in Table 7.3.

Table 7.3 confirms that the step size of the fitness change for our new crossover operators (SSC, SASE and MSSC) was smaller than for either NSM, SAC or SC, and thus the change in fitness over the generations of SSC, SASE and MSSC was smoother than for SAC, NSM and SC. By making a semantically small change in the parents as in SSC, SASE and MSSC one can achieve a small change in semantics of their offsprings. This result is important, as it is not trivial to achieve the locality property by making syntactically small changes. This not only helps to improve the performance of GP as shown in the previous chapters, but may also provide an impetus to attract other GP researchers to find more effective ways of achieving locality for their GP systems. The table also shows that the fitness change of both SAC and NSM was only slightly smoother than SC on some problems. This can be seen as a reason why SAC and NSM were not significantly better than SC as shown in the previous chapters. Comparing between the two new improved versions of SSC, it can

be seen that the step size of the fitness change of SASE is mostly the same with one of SSC with two exceptions on function F_7 and F_8, whereas MSSC often make smaller change than both SSC and SASE.

7.4 Constructive Effects

The results from the previous sections show that SAC, SSC, SASE and MSSC are more semantically productive than SC, and that SSC, SASE and MSSC have higher locality than both SAC and SC. This leads to a further question, whether these properties help the crossover operators to generate better children than their parents (more constructive crossover). In other words, we would like to know the relative constructiveness of the operators, SSC, SASE, MSSC, SAC and SC. Statistics were collected measuring the constructive effect of NSM, SAC, SSC, SASE, MSSC and SC, using a method similar to that in [117]. The constructive effect is measured by calculating the percentage of crossover events that generate children better than their parents.

We distinguish two categories of constructive crossovers: *semi-constructive crossovers* and *full-constructive crossovers*. Let us assume that two parents P_1 and P_2 are selected for crossover, generating two children C_1, C_2 (C_1 rooted at P_2, and C_2 rooted at P_2). Then, a crossover is called semi-constructive if it generates at least one child that is better then its parents. In other words, the condition (C_1 *is better than* P_1 *OR* C_2 *is better than* P_2) is used to count semi-constructive crossovers. When the condition is more strict – both children are better than their parents (C_1 *is better than* P_1 *AND* C_2 *is better than* P_2) – the crossover is called full-constructive. A crossover that is not semi-constructive nor full-constructive is called destructive.

The semi-constructive and full-constructive crossovers results for these operators are shown in Table 7.4 and Table 7.5, respectively. It can be seen from Table 7.4 that both methods that promote semantic diversity, NSM and SAC were more semi-constructive than SC. This is a consequence of the greater semantic diversity of SAC and NSM relative to SC. The table also shows that while NSM is only marginally better than SC, from 1% to

Tab. 7.4: The percentage of semi-constructive crossovers of SC, NSM, SAC, and SSC, SASE, MSSC (i.e. at least one child is better than the corresponding parent).

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	21.2	21.3	20.8	19.0	24.9	19.2	31.1	31.2
NSM	23.4	22.6	23.2	21.1	24.7	21.4	32.6	31.9
SAC02	28.1	27.3	29.1	23.0	30.0	25.8	37.7	37.0
SAC03	27.8	28.5	28.3	23.6	29.9	26.1	36.9	37.0
SAC04	27.9	27.9	28.0	24.3	31.0	26.4	37.0	37.0
SSC12	34.3	35.2	33.5	31.5	38.8	32.2	40.9	41.9
SSC16	36.0	35.3	34.8	32.8	39.1	32.7	40.8	42.3
SSC20	36.8	36.1	34.7	32.4	40.2	34.1	42.1	42.2
SASE65	34.7	34.8	33.9	31.0	38.2	31.6	40.5	40.7
SASE75	35.7	35.1	34.4	30.9	37.8	33.2	40.7	41.9
SASE85	36.3	34.9	34.4	31.7	38.8	33.6	40.1	42.0
SASES	35.4	35.4	35.6	31.9	39.7	33.6	41.0	42.2
MSSC12	37.1	38.6	37.6	35.0	38.9	35.2	43.3	43.9
MSSC16	40.0	40.3	39.6	37.0	39.9	36.7	45.8	46.2
MSSC20	40.5	41.7	41.8	37.5	41.5	38.5	47.3	46.9

3%, SAC is often better than both SC and NSM, from 5% to 8%. This can be seen as the result of more semantic exploration of SAC versus NSM and SC on these functions. These increases are particularly important because the semi-constructive rate for SC was rather small (about 20%).

While promoting semantic diversity like SAC helps to enhance the probability of crossover that generates better children, improving semantic locality as in SSC is even more effective. The tables shows that SSC creates the probability of better children to a greater extent. Usually, SSC is often from 10% to 15% more semi-constructive than SC. Comparing between two improved methods of SSC, it can be observed that SASE is roughly equal to SSC in terms of semi-contructive effect, while MSSC is often 5% better than both SSC and SASE. These results provides support to explain why the performance of MSSC is often better than SSC, while the better performance of SASE can be the result of the balance between local and global search.

Table 7.5 shows how difficult it is for standard GP crossover to generate improved

Tab. 7.5: The percentage of full-constructive crossovers of SC, NSM, SAC, and SSC, SASE, MSSC (i.e. both children are better than the corresponding parent).

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	2.27	2.25	2.28	2.00	2.62	1.97	4.20	4.02
NSM	2.49	2.39	2.52	2.19	2.77	2.14	4.32	4.23
SAC2	3.28	3.25	3.56	2.62	3.18	2.87	5.42	5.18
SAC3	3.29	3.43	3.45	2.63	3.80	2.96	5.26	5.17
SAC4	3.31	3.32	3.40	2.78	3.94	2.96	5.28	5.17
SSC12	4.94	5.15	4.58	4.07	5.40	4.52	6.21	6.29
SSC16	5.25	5.15	4.80	4.36	5.45	4.57	6.30	6.31
SSC20	5.33	5.30	4.69	4.63	5.71	5.00	6.29	6.13
SASE65	5.07	5.06	4.81	4.19	5.39	4.21	5.91	6.07
SASE75	5.01	5.09	4.84	4.11	5.25	4.37	6.01	6.26
SASE85	5.22	4.98	4.86	4.17	5.31	4.57	6.06	6.37
SASES	5.09	5.13	5.12	4.33	5.41	4.67	5.12	6.34
MSSC12	5.55	5.80	5.49	4.84	5.41	5.05	6.51	6.47
MSSC16	6.08	6.13	5.66	5.34	5.49	5.39	6.40	6.81
MSSC20	5.99	6.24	5.85	5.47	5.69	5.66	6.49	6.63

solutions. The percentage of fully constructive crossovers for SC was only about 2% for one-variable functions and 4% for bivariate functions. But adding semantics to crossover enabled many more full-constructive crossovers. SAC often scored 1.5 times higher than SC in frequency of full-constructive events, and SSC scored around 2 times higher. This table again shows that SSC and SASE are mostly equal while MSSC is always the best crossover in terms of full-constructive effect.

7.5 Code Bloat Effect

This section is dedicated to the investigation of the (side)effect of promoting semantic diversity and improving semantic locality of crossovers on code bloat in GP. It starts with a comparison of the effect of the above crossovers on the code bloat. Then, the relationship between semantic locality of crossovers and bloat is studied.

Tab. 7.6: Average size of individuals over all generations (i.e. the number of nodes in each individuals).

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	53.5	50.7	48.9	60.8	48.6	57.4	42.8	41.7
NSM	52.8	51.1	47.1	63.2	48.9	57.9	43.5	41.6
SAC02	56.6	54.9	48.6	64.7	51.4	58.2	44.8	45.9
SAC03	57.2	55.3	48.4	63.6	50.7	59.7	46.8	44.9
SAC04	57.8	56.1	48.2	64.4	51.2	58.5	46.6	44.7
SSC12	47.0	46.6	46.5	60.1	46.3	51.0	41.6	40.2
SSC16	46.9	47.1	46.0	59.7	48.2	50.2	41.5	40.4
SSC20	47.0	48.0	46.8	58.6	48.1	49.6	41.4	40.3
SASE65	45.7	47.0	43.2	59.8	47.3	53.1	23.0	27.7
SASE75	44.6	46.2	44.7	59.3	48.2	53.2	27.6	29.5
SASE85	48.0	48.8	42.5	59.7	44.9	51.8	33.2	34.0
SASES	45.4	45.8	43.6	59.4	45.3	52.9	26.0	28.4
MSSC12	42.2	42.1	41.5	57.7	46.7	50.7	25.7	33.8
MSSC16	40.9	41.6	40.6	54.0	46.0	48.5	21.5	27.9
MSSC20	41.1	40.8	40.5	55.2	45.4	44.7	20.7	28.7

7.5.1 Semantic Diversity, Semantic Locality and Code Bloat

It has been known since the early days of GP that the average size of programs inexorably grows, sometimes even exponentially [9]. This phenomenon is known as code bloat [2].

There are a number of explanations for code bloat in the GP literature (e.g. [109, 143, 177, 38]. Although this is still an area of debate, it is generally agreed that code bloat negatively impacts GP search, and also makes GP solutions difficult to comprehend. While a comprehensive study of the role of semantics on GP code bloat is not the main objective of this thesis, it is interesting to see how semantic based crossover affects GP code bloat.

To investigate the effect of the semantic based crossovers on code bloat we measured the average size of individuals (number of nodes) over 50 generations, averaged over 100 runs. The statistics are presented in Table 11.7. The table reveals that promoting semantic diversity as in SAC and NSM often leads to a little more bloat than SC. The table also shows that SAC often exhibits more bloat than NSM. The reason could be that SAC is

[2]In this thesis, we use terms *code bloat* simply means the growth of population size.

Tab. 7.7: Average size of the solutions.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	66.4	73.0	53.3	78.1	67.4	67.1	63.9	65.4
SSC12	59.3	70.8	58.7	83.8	60.9	55.3	62.4	51.8
SSC16	60.1	47.0	48.0	86.7	65.6	51.8	55.8	55.6
SSC18	65.6	64.5	50.7	79.8	76.6	45.5	58.6	59.9
SASE65	65.9	67.3	52.0	75.5	66.8	56.7	38.5	41.9
SCAS75	61.6	59.4	47.3	83.9	73.5	58.3	47.3	41.1
SASE85	60.5	47.5	50.7	74.6	69.6	63.8	49.1	49.8
SASES	51.5	56.4	50.0	86.4	62.3	63.9	41.9	41.3
MSSC12	51.5	50.3	52.5	75.0	62.9	55.3	38.1	39.2
MSSC16	50.2	63.4	44.5	55.6	63.7	53.8	29.0	28.3
MSSC20	50.1	34.6	43.7	77.3	65.8	53.4	25.5	24.6

more semantically exploratory than NSM. However, it also indicates that the amount of excessive bloat was acceptable. On the contrary, improving semantic locality as in SSC usually reduces code bloat compared to SC. Similarly, both new improvements of SSC, SASE and MSSC have less bloat than SC, and MSSC is the best method in terms of reducing code bloat.

The reduction in code bloat of SSC, SASE and MSSC over SC, consequently, reduces the running time. The results in Chapter 6 show that SSC, SASE, especially MSSC, usually run faster than SC. The exception only lies in some cases of SSC. Generally, the average running time of SASE and MSSC was only equal to 80% the average running time of SC. In the group of bi-variable functions, the running time, compared to SC, was often reduced a half with MSSC (see Chapter 6).

Another benefit of having less bloat in SSC, SASE and MSSC is that it potentially helps to find more comprehensible solutions (smaller and more readable solutions). This is confirmed by the result in Table 7.7, where we show the average size of the solutions found in the runs. It can be seen from the table that SSC, SASE and MSSC usually found solutions with better quality (in terms of size). The exception again lies in function F_3, F_4 and F_5 with SSC and SASE. In the comparison among SSC, SASE and MSSC, the result is consistent with the previous results where MSSC was usually better than SSC and SASE

in terms of finding smaller solutions. Overall, the improvement of SSC, SASE and MSSC over SC were not only in performance but also in reducing code bloat, which led to smaller running times and shorter solutions.

7.5.2 The Relationship of Semantic Locality and Code Bloat

The previous subsection shows that improving semantic locality leads to reduced code bloat in GP. It also seems that higher semantic locality of crossover tends to even further reduce code bloat. To investigate the relationship between semantic locality and code bloat we designed an experiment that tries to measure this correlation by tunning the semantic locality of these crossovers (by changing semantic sensitivities) from low to high. The experimental settings are as follows:

For SSC, to increase its semantic locality, we need to reduce the upper bound semantic sensitivity. It should be noted that an upper bound of semantic sensitivity that is too small could deter SSC performance. However, since the objective of the experiment was purely to focus on code bloat, the performance was ignored. Five values for the upper bound of semantic sensitivity were examined: 0.05, 0.1, 0.2, 0.4, and 0.8. As reducing upper semantic sensitivity, will make it more difficult to select a pair of subtrees that satisfy SSC's condition, the number of trials (MT) needs to be increased. Therefore, the values of MT for the above semantic sensitivities were 80, 40, 20, 10, and 5 respectively. In total, 5 configurations of SSC were tested and will be referrer to as SSCX with X=5, 10, 20, 40, 80 respectively.

For SASE based crossover(s), SASES was used in our experiment. It can be seen that the semantic locality of both SASE and MSSC will be higher when we increase the number of subtree selection trials (MT). Therefore, similar to SSC, 5 values of MT: 5, 10, 20, 40, 80, were set for each SASE and MSSC. They comprise 5 configurations for SASE and for MSSC and are called SASEX and MSSCX with X=5, 10, 20, 40, 80 respectively. To investiage the relationship between semantic locality and code bloat we also measured the average size of individuals (number of nodes) over 50 generations, averaged over 100 runs.

Tab. 7.8: Average size of individuals over all generations (i.e. the number of nodes in each individuals) when increasing semantic locality of crossovers.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC	53.5	50.7	48.9	60.8	48.6	57.4	42.8	41.7
SSC5	51.1	50.4	48.7	60.3	47.5	53.2	44.7	42.1
SSC10	46.1	49.4	45.0	60.0	46.6	49.4	42.2	41.3
SSC20	45.1	45.0	42.5	60.0	45.4	49.1	30.7	34.4
SSC40	43.2	43.5	43.7	59.3	46.1	48.2	25.8	31.6
SSC80	46.7	45.3	46.1	62.3	48.6	52.5	27.8	32.4
SASE5	49.1	49.2	46.4	61.4	49.6	54.8	38.3	36.4
SASE10	45.2	47.8	43.9	61.2	48.2	51.6	28.9	34.5
SASE20	42.8	46.5	41.2	59.9	47.1	49.3	21.2	26.6
SASE40	42.2	43.9	43.0	59.0	47.0	47.9	21.0	26.1
SASE80	46.3	48.1	45.4	62.3	49.8	53.7	24.0	27.6
MSSC5	49.3	50.8	47.7	61.9	47.9	53.8	38.3	39.7
MSSC10	43.9	47.6	45.1	59.2	46.8	51.6	28.3	35.7
MSSC20	41.1	40.8	40.5	55.2	45.4	44.7	20.7	28.7
MSSC40	40.4	39.8	38.7	54.6	39.8	42.6	20.3	26.0
MSSC80	40.7	42.1	42.3	53.9	41.5	44.0	20.9	25.8

The results are presented in Table 7.8.

It can be observed from this table that there was a strong correlation between semantic locality and code bloat where increasing the semantic locality of crossovers led to further reduction in code bloat. The table shows that the average size of individuals is consistently decreased when MT was increased from 5 to 40. It is interesting, however, that when the semantic locality went higher it did not help to further reduce code bloat. The average individual size when MT is 80 was often larger than when MT is 40. Overall, the results in this section show that promoting semantic diversity of crossovers tends to increase code bloat while improving crossover semantic locality leads to a remarkable reduction in code bloat. Moreover, the results also confirm that the higher semantic locality of crossover, the less problem with code bloat GP will have. However, this locality should not be arbitrarily high as there is a saturation point where higher semantic locality of the crossover operator could not help to reduce code bloat only to require more running time.

7.6 Semantics Exchanged in Crossovers

The previous sections showed a number of advantages of improving semantic locality of standard crossover, but why should we improve semantic locality of standard crossover? This section addresses that question by analysing the semantic exchanged in SC and SSC. SAC is not investigated here since it could be seen as a special version of SSC with the upper semantic sensitivity set to an extreme value. The section starts with some analysis on the semantic exchanged between two subtrees in SC. The effects of this semantic exchange on the semantic locality and constructive effect are also examined. Then, the semantic exchange of two subtrees of SSC is given.

7.6.1 Semantics Exchanged in Standard Crossovers

As SSC works by both preventing the exchange of two semantic equivalent subtrees and keeping a small semantic difference, it is informative and useful to see semantics of two subtrees exchanged by this crossover compared to SC. In order to investigate this we designed an experiment as follows:

Let Sem be the semantic distance of two subtrees exchanged in SC. We divide SC into 7 groups, these 7 groups will be referred as SCs from here, as follows:

1. SC0 if $Sem < 10^{-3}$
2. SC1 if $10^{-3} \leq Sem < 0.1$
3. SC2 if $0.1 \leq Sem < 0.2$
4. SC3 if $0.2 \leq Sem < 0.4$
5. SC4 if $0.4 \leq Sem < 0.8$
6. SC5 if $0.8 \leq Sem < 1.6$
7. SC6 if $1.6 \leq Sem$

Tab. 7.9: The percentage of each groups in 7 groups of Standard Crossover.

Methods	F1	F2	F3	F4	F5	F6	F7	F8
SC0	14.3	14.6	17.3	13.2	14.1	14.4	11.8	12.2
SC1	2.65	2.68	4.63	0.96	1.14	0.60	4.63	5.40
SC2	3.23	3.13	5.48	3.73	3.68	2.42	4.80	4.38
SC3	4.70	4.54	10.7	11.6	6.34	2.88	28.5	24.7
SC4	22.1	22.3	31.6	25.5	11.7	11.2	28.8	30.0
SC5	38.3	38.4	17.7	24.5	33.9	35.0	14.4	15.9
SC6	14.7	14.2	12.4	20.4	29.0	33.3	6.41	6.79

SC0 corresponds to the situation where SC exchanges two semantically equivalent subtrees. SC1, SC2 and SC3 are instances where we expect SSC to operate most effectively and three remaining cases (SC4, SC5 and SC6) are the situations when SC exchanges subtrees which are semantically too different. We firstly recorded how many operations of SC that happened during our runs of GP falling in each of the seven groups at each generation. The values are then averaged over 50 generations and 100 runs and are shown in the Table 7.9.

It could be seen from Table 7.9 that there was a small portion of SC events that exchange two semantically similar subtrees (with Sem lies in the interval from 10^{-3} to 0.4). In total, there was only 10% to 15% of SC that lies in these three cases. In terms of exchanging two semantic equivalent subtrees, the results show from 12% to 15% of SC. It is not a large percentage but it also reduces the semantic diversity of SC that will usually lead to a poorer performance of SC compared to SAC and other crossover operators for promoting semantic diversity. By contrast, there was a large number of times SC exchanged two semantically dissimilar subtrees. The table shows that there was from 50% to 70% of SC operations that swapped two subtrees with their semantic distance greater than 0.4. Therefore, if the exchange of two semantic dissimilar subtrees causes an abrupt movement of semantics during the course of evolution, preventing this situation is essentially important to smooth out the change of semantics.

To investigate the effect of semantic exchange between two subtrees on the semantic

Tab. 7.10: The change of semantics from parents to children in Standard Crossover.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SC1	0.23	0.27	0.16	0.06	0.08	0.09	0.07	0.04
SC2	0.20	0.20	0.23	0.11	0.15	0.16	0.08	0.07
SC3	0.45	0.37	0.28	0.21	0.19	0.36	0.13	0.13
SC4	0.56	0.57	0.47	0.30	0.27	0.58	0.20	0.21
SC5	0.72	0.89	0.71	0.46	0.44	0.67	0.33	0.43
SC6	2.02	2.20	1.53	0.72	1.25	1.39	1.02	1.07

Tab. 7.11: Full-constructive events of Standard Crossover.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SC1	9.58	9.66	8.38	7.79	9.19	8.58	11.9	12.2
SC2	6.72	7.02	5.06	5.39	7.73	6.26	7.41	7.38
SC3	4.51	4.56	3.96	3.84	5.75	4.13	5.81	5.70
SC4	2.72	2.77	2.60	2.59	4.14	2.92	4.29	4.21
SC5	1.99	2.00	1.42	1.60	3.10	2.17	2.94	2.90
SC6	1.49	1.41	1.23	1.30	1.95	1.61	2.30	2.21

movement from the parents to the children, we recorded semantic distance of two consecutive populations. The value is then averaged over 50 generations and 100 runs. The results are shown in Table 7.10. It can be seen from this table that when SC exchanges two semantic equivalent subtrees, the semantics transfered from parents to children is likely unchanged. Therefore, there was at least from 12% to 15% of SC operation that will not change the semantics of individuals. Moreover, when SC exchanges two semantically similar subtrees, the change of semantics from parents to children is often much smoother than when it exchanges two semantically dissimilar subtrees. It can be observed that the population semantic distances of SC1, SC2 and SC3 were usually much smaller than for SC4, SC5 and SC6. Hence, we argue that making a semantically small change leads to a smooth movement of semantics during the course of evolution, while a big change leads to an abrupt movement.

What is more important is that a smooth semantic movement leads to a more construc-

tive effect of crossovers while an abrupt semantic movement leads to a less constructive effect. It is confirmed by the results in Table 7.11 where the percentage of full-constructive events with each kind of SC is shown. Here, the concept of full-constructive crossover is similar to the concept in the previous section. In other words, assume that two parents P_1 and P_2 selected for crossover generates two children C_1, C_2 (C_1 rooted in P_1 and C_2 rooted in P_2), then, a crossover is called constructive if C_1 *is better than* P_1 *AND* C_2 *is better than* P_2.

We counted the percentage of full-constructive events of seven classes of SC at each generation. The values are then averaged for all generations and over 100 runs and shown in Table 7.11. It can be seen from this table that SC0 does not change the semantics of individuals, therefore, can not improve their fitness. Moreover, making a semantically smooth change led to a much greater full-constructive effect than when making a semantic abrupt change. The percentage of full-constructive events of SC1, SC2 and SC3 was from 4% to 12% while these values of SC4 was only around 2% to 3% and of SC5 and SC6 were event smaller with only approximately 1% to 2%. To summarize, this subsection shows that most of SC operations (events) exchanges two semantic equivalent subtrees or two semantic dissimilar subtrees. Furthermore, exchanging two semantic similar subtrees leads to a smoother change of semantics from parents to children while exchanging two semantic dissimilar subtrees leads to an abrupt change of semantics. This, consequently, leads to a greater full-constructive effect of exchanging two semantic similar subtrees versus exchanging two semantic dissimilar subtrees.

7.6.2 Semantics Exchanged in Semantic based Crossovers

The previous subsection shows that there is only a tiny portion of SC events that exchange two semantic similar subtrees. Moreover, SSC is aims to increase the exchange of two semantic similar subtrees, it is important to see how SSC is successful in this objective. In order to investigate, we designed an experiment that is similar to the previous experiment with SC. As before, the upper bound of semantic sensitivity of SSC in this experiment was

Tab. 7.12: The percentage of each groups in 7 groups of SSC.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SSC0	3.17	3.36	3.10	1.79	2.57	4.74	0.29	0.42
SSC1	24.2	24.7	24.6	6.40	11.5	8.64	11.3	14.5
SSC2	26.0	24.7	23.7	21.3	28.8	26.8	12.6	12.7
SSC3	30.8	30.5	41.5	59.9	43.7	30.6	74.6	70.9
SSC4	4.60	4.74	3.42	3.89	1.99	4.27	0.41	0.56
SSC5	8.17	8.62	1.94	3.46	5.77	12.6	0.38	0.49
SSC6	2.91	3.25	1.58	3.17	5.43	12.1	0.31	0.40

set as 0.4 and Max_Trial is set at 12. Similar to SC, SSC is divided into seven groups. These seven groups will be referred to as SSCX (X=0, 1,..., 6). We recored the percentage of SSC that lies in each of the seven groups at each generation. The values are averaged over 50 generations and 100 runs. The results are shown in Table 7.12.

It can be seen from this table that the portion of SSC that semantically exchanges two similar subtrees is substantially increased compared to SC. There is about 80% to 90% of SSC that lies in SSC1, SSC2 or SSC3. At the same time, the percentage of SSC that exchanges two semantic equivalent subtrees and the percentage of SSC that swaps two semantic dissimilar subtrees are remarkably reduced. This table provides the evidence that SSC actually achieved its design objective to keep a small change of parents in crossover in terms of semantics.

We also recorded this statistic for new improved methods of SSC, SASE and MSSC. The results, although are not shown in this chapter (See Appendix A), show that both SASE and MSSC achieved their design objective in terms of keeping a small semantic change from parents to children.

7.7 Conclusion

This chapter investigated some properties of semantic based crossovers. The comparison was undertaken with standard crossover and a fitness-based method, SNM. The results

showed that there is about 12% to 15% of SC that exchanges two semantically equivalent subtrees and the semantic based crossovers are successful in preventing this phenomena. The results also show that semantic based crossovers are more exploratory than SC and that crossovers that aim to keep a small change from parents to children enhance their semantic locality. This results in a more constructive effect of semantic based crossover in comparison with the left crossovers, meaning that semantic based crossovers are more likely to produce children that are fitter than their parents.

The effect of the crossovers under semantic control on GP code bloat was also examined and the results showed that promoting semantic diversity leads to more bloat than SC. However, the amount of excessive bloat is small and acceptable. On the contrary, improving semantic locality helps to further reduce code bloat and this helps these operators run faster than SC. The quality of the solutions that are found by these crossovers are also better in terms of size of the solution.

Finally, a comprehensive analyses of semantics exchanged of SC and SSC was conducted and the results show that there is about 12% to 15% of SC events that do not change semantics of parents and a large portion of SC (50% to 70%) that exchanges two semantic dissimilar subtrees. This leads to a low value of semantic locality of SC. Low semantic locality results in abrupt movements of semantics during the course of evolution and this in turn leads to the destructive effect of SC. The analysis of SSC showed that SSC helps to solve problem of low semantic diversity and especially semantic locality of SC. We argued that this is the main reason for the better performance of SSC and its two new improved versions, SASE and MSSC. The next chapter will investigate the generalisation ability of semantic based crossover, SSC and MSSC (SASE is not investigated more as its behaviour is similar to SSC).

Chapter 8

Examining Generalisation Ability of Semantic based Crossovers

The previous chapters have shown that semantic based crossovers improve the performance of GP on training data. This chapter examines the generalisation ability of GP using these semantic based crossovers. First, a review of work on GP generalisation ability is given. Then we present the GP parameter settings and the problems that are used to test the generalisation ability of GP. Next, the generalisation ability of the semantic based crossovers, Semantic Similarity based Crossover (SSC) and the Most Semantic Similarity based Crossover (MSSC) are compared with the standard crossover, the validation set method and two bloat control methods including a multi-objective approach and Tarpeian Bloat Control. Some preliminary results in this chapter have been published in [135].

8.1 Introduction

In the field of Machine Learning (ML), generalisation has been seen as one of the most desirable properties for learning machines [125]. Since GP could be seen as a (evolutionary) machine learning methodology, it is very important to guarantee that the solutions GP finds, not only work well on training data but also on the unseen data [29]. Surprisingly,

a large number of GP researchers have only reported results on training data. While overfitting the training data to get the exact solutions is suitable in some cases, for most learning problems in reality it would not be enough without considering their generalisation over unseen data.

Some recent research (e.g. [29, 182, 52]) has shown that the ability of GP to generalise could be poor. The awareness of the ability of GP to generalise is also important in the context of performance comparison between different GP systems. It has been recently shown [29] that an enhanced GP system performance might be remarkably better than standard GP on training data, but not significantly better on unseen data.

The previous research on improving the ability of GP to generalise is mostly focused on reducing the solution size [182, 52, 115, 42]. The motivation for such an approach is that GP usually bloats, with solution complexity (size) increasing rapidly during the evolutionary process. High complexity solutions are often poor in their ability to generalise as they contradict Ockham's razor principle [125] (simple solutions are prefered). In this chapter, we demonstrate the ability to improve generalisation of GP using a semantic based approach. In particular, we test if the previously proposed semantics based crossovers, namely the Semantic Similarity based Crossover (SSC) and the Most Semantic Similarity based Crossover (MSSC) could improve the ability of GP to generalise. The experimental results show the effectiveness of the SSC and MSSC approaches in comparison with standard GP, the validation set based method and the bloat reduced methods.

8.2 A Review of Generalisation in Genetic Programming

Although generalisation of learned solutions is the primary interest of any learning machine [125], it was not seriously considered in the field of GP for a long time. Before Kushchu published his work on the generalisation ability of GP [105], there was little research dealing with the GP generalisation aspect. Francone et al. [49] proposed a new GP

system called Compiling GP (CGP) and the authors compared its generalisation ability with that of other machine learning techniques. The results show that the ability of CGP to generalise compares favourably with a number of more traditional machine learning methods. Furthermore, the influence of using extensive mutation on the ablity of CGP to generalise was investigated and the experimental results show positive effects [10]. Ekart and Nemeth [42] tested the generalisation ability of GP that is based on a multi-objective method. The results show that the multi-objective method based on Pareto Nondomination Criterion help to reduce code bloat while maintaining the the ability of GP to generalise.

Recently, the issue of generalisation in GP is deservedly receiving increased attention. Mahler et al. [115] experimented with Tarpeian Control on some symbolic regression problems and tested the side effects of this method on the generalisation ability of GP. The results were inconsistent and problem dependent, i.e., it can either increase or reduce the generalisation power of solutions found by GP. Gagné et al. [52] investigated two methods to improve generalisation in GP-based learning: the selection of the best of run individuals using a three datasets method (training, validation, and test sets), and the application of parsimony pressure in order to reduce the size of the solutions. Their experimental results indicate that using a validation set could slightly improve the stability of the best of run solutions on the test sets. Costa et al. [28] proposed a new GP system called relaxed Genetic Programming with generalisation ability better than standard GP.

More recently, Costelloe and Ryan [29] showed the important role of generalisation on GP. They experimentally showed that a technique like Linear Scaling [92] may only be significantly better than standard GP on training data but not superior on testing data. They proposed an approach to improve GP generalisation by combining Linear Scaling and the No Same Mate strategy [62]. Vanneschi and Gustafson [182] improved GP generalisation using a crossover based similarity measure. Their method is to keep a list of over-fitting individuals and to prevent any individual entering the next generation if it is similar (based on structural distance or a subtree crossover based similarity measure) to one individual in the list. The method was then tested on a real-life drug discovery regression

Tab. 8.1: Symbolic Regression Functions for investigating GP generalisation ability.

Functions	Training Data	Testing Data
$F_1 = x^4 + x^3 + x^2 + x$	30 random points $\subseteq$ [-1,1]	100 $\subseteq$[-1:0.02:1]
$F_2 = x^3 - x^2 - x - 1$	60 random points $\subseteq$ [-1,1]	100 $\subseteq$[-1:0.02:1]
$F_3 = arcsin(x)$	30 random points $\subseteq$ [-1,1]	200 $\subseteq$[-1:0.01:1]
$F_4 = \sqrt{x}$	60 random points $\subseteq$ [0,4]	200 $\subseteq$[0:0.02:4]
$F_5 = 0.3sin(2\pi x)$	30 random points $\subseteq$ [-1,1]	100 $\subseteq$[-0.5:0.02:1.5]
$F_6 = cos(3x)$	60 random points $\subseteq$ [-1,1]	200 $\subseteq$[0:0.01:2]
$F_7 = xy + sin((x-1)(y+1))$	60 random points $\subseteq$ [-1,1]	100 $\subseteq$[-1:0.02:1]
$F_8 = x^4 - x^3 + y^2/2 - y$	60 random points $\subseteq$ [-1,1]	100 $\subseteq$[-1:0.02:1]

problem and the experimental results showed improvements on the ability to generalise. Castelliet et al. [22] compared the generalisation ability of different GP frameworks. The results showed that a multi-objective method is effective in improving GP generalization ability in solving a hard regression real-life application in the field of drug discovery and development. The authors then further investigated the relationship between generalisation ablity and solutions functional complexity [186, 23] and their experimental results show that there is a correlation between the generalisation ability of GP solutions and their functional complexity measured by Graph Based Complexity. It should be noted here that, most research on improving the ability of GP to generalise has been purely focused on reducing the complexity of the solution and semantic control has not been considered as an approach to enhance ability of GP to generalise.

8.3 Experimental Settings

This section presents the problems used to test generalisation ability of semantic based crossovers and the parameter settings for tested methods.

8.3.1 Symbolic Regression Problems

To investigate the impact of semantic based crossovers on the ability of GP to generalise, we used eight real-valued symbolic regression problems. The tested problems, training and

testing data are shown in Table 8.1. These functions were taken from some other work on GP learning generalisation [29, 48, 115, 92]. The training sets are used to train the GP system. The best individual in terms of fitness of each run is selected. This individual is tested on the test sets to determine the generalisation ability of GP solutions. It is noted that the testing sets are often much larger than the training sets and in some cases they contain values that are not in the training intervals (F_5, F_6). This makes the experimental setting more general.

8.3.2 Parameter Settings

The basic experimental settings used in this chapter are as in Table 5.2. The ability of semantic based crossovers (SSC and MSSC) to generalise is compared with a validation set method [52], Tarpeian Bloat Control [155, 115], a multi-objective method [42]. The configuration and naming convention for SSC and MSSC are similar to those adopted in Chapter 6.

For the validation set method (referred to as VAL method) the training set is randomly divided into 2 (for each run): 67% is used for training (training set) and the remaining 33% is used for validating (validation set). At each generation the fitness of individuals is measured on the training set and this fitness is used for tournament selection. At the same time, a two-objective trial (fitness and size of an individual) is conducted in order to extract a set of non-dominated individuals (the Pareto front). The individuals in the Pareto front are then evaluated on the validation set, with the best of run individual selected as the one of these with the smallest error rate on the validation set. This configuration is similar to the validation configuration in [52].

The multi-objective method [42] is shorthanded as MUL. The idea of this approach is to select an individual for the mating pool if it is not dominated by any individuals from a previous selected individuals set. Here individual gp_1 dominates individual gp_2 if $fitness(gp_1) < fitness(gp_2)$ and $size(gp_1) < size(gp_2) + bias$. Although, there are three different ways to determine $bias$ that affect the size of the selected individual [42], here

we use the first way that assigns *bias* to a threshold. It has been shown [42] that this is an effective way to reduce code bloat while still maintain generalisation ability. Three different threshold, 20, 30, 40, will be used. These three configurations of MUL will be referred as MULX with X=20, 30, 40 in the following section.

The last method is Tarpeian Bloat Control (TBC) [155, 115]. The idea behind Tarpeian Bloat Control is to assign a very low fitness for the individuals that have a size greater than the average size of the population when a random generated number is smaller than the *Target Ratio*. In this experiment, three value of Target Ratio, 0.1, 0.2, 0.3, are used. These three configurations of TBC is denoted as TBCX with X=01, 02, 03.

8.4 Results and Dicussion

This section investigates the ability of semantic based crossovers (SSC and MSSC) to generalise. First, we compare the GP performance of semantic based crossovers with other methods. The impact of these methods on GP code bloat and on the ability of GP to generalise are presented after that.

8.4.1 On the Performance of GP

Since there could be a relationship between the error of the solutions on training sets and on testing sets, we first compare the performance of GP of these methods on the training sets. We recorded a classic performance metric, the mean best fitness of these methods on the training data. The results are shown in Table 8.2. This table is consistent with the results in Chapter 6, in which both SSC and MSSC help to remarkably improve GP performance. We also statistically tested the improvement of SSC and MSSC versus SC in Table 8.2 using Wilcoxon signed-rank test with a confidence level of 95%, and the results of the statistical test show that both SSC and MSSC always significantly improve the performance of GP in comparison with SC. The results in this table are also consistent with those in Chapter 6 where it again confirms that MSSC is often better than SSC.

Tab. 8.2: The average of best fitness on training set. Note that the values are scaled by 10^2.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.54	3.42	0.49	1.40	2.45	1.34	11.1	13.3
VAL	1.27	2.87	0.53	1.18	2.27	1.46	10.64	12.8
MUL20	2.57	4.57	1.76	1.50	4.77	1.86	13.4	15.1
MUL30	2.31	3.39	1.97	1.77	4.26	1.74	12.1	14.8
MUL40	2.39	3.61	1.88	1.50	3.73	1.80	11.9	13.8
TBC01	1.32	3.44	0.74	1.31	2.69	1.57	12.6	13.4
TBC02	1.60	3.93	0.62	1.46	2.99	1.63	13.0	13.9
TBC03	3.38	4.57	1.24	1.82	3.50	1.82	15.4	16.8
SSC12	0.85	1.85	0.28	0.62	2.11	0.92	8.59	9.89
SSC16	0.82	1.52	0.29	0.67	1.89	0.82	8.56	9.92
SSC20	0.86	1.53	0.30	0.71	1.87	0.74	8.63	9.24
MSSC12	0.76	1.28	0.22	0.53	1.77	0.79	8.47	8.43
MSSC16	0.77	1.40	0.21	0.55	1.77	0.46	9.02	8.60
MSSC20	0.82	1.45	0.20	0.66	1.88	0.90	9.35	9.91

With other methods, we can see that there is a very limited improvement. While there is a slight improvement of VAL on some problems (however, Wilcoxon signed rank test confirms it is not significant), there is almost no improvement of MUL and TBC in terms of GP performance in comparison with SC. These results are understandable since these methods do not aim to improve the performance of GP on the training data.

8.4.2 On the Code Bloat Effect

Similarly, there could be a strong correlation between the complexity of solutions and their ability to generalise (Ockham's razor or Minimum Description Length - MDL principle [125]), statistics on solution size were also recorded and analysed. This includes the average size of the population averaged over all generations and the average size of the best fitness on the training data (with validation set method it is the average size of the best fitness on the validation data) by each method. These results are depicted in Table 8.3 and Table 8.4, respectively.

Tab. 8.3: The average of population size.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	52.4	60.5	46.1	55.8	70.5	54.5	50.6	48.1
VAL	50.0	57.6	42.8	51.7	65.7	55.8	49.1	55.8
MUL20	23.5	32.6	26.4	30.4	35.2	28.7	28.6	27.4
MUL30	30.2	40.5	28.9	38.2	39.2	37.0	33.8	32.4
MUL40	37.8	47.5	32.0	30.4	47.0	43.4	39.2	37.1
TBC01	39.9	50.8	35.5	44.9	53.9	46.8	40.1	37.8
TBC02	26.7	38.0	23.7	34.1	40.8	35.4	31.1	29.4
TBC03	12.0	26.6	12.2	24.8	27.4	26.1	21.8	20.6
SSC12	47.3	57.9	43.9	49.7	69.3	45.2	46.7	42.2
SSC16	44.4	57.6	42.9	49.5	64.5	44.4	44.5	41.0
SSC20	46.4	56.7	42.8	49.2	64.9	42.8	43.6	38.1
MSSC12	40.5	52.8	39.8	46.8	65.4	43.9	41.1	38.0
MSSC16	41.3	49.9	38.4	45.0	65.4	39.7	40.2	35.2
MSSC20	38.0	48.9	38.4	44.6	62.8	41.0	36.5	34.8

It can be seen from Table 8.3 that all these methods help to reduce GP code bloat, however the extent in which they reduce code bloat is different. Obviously, VAL only marginally reduce bloat, while MUL and TBC reduces GP code bloat to a greater extent. Usually, the average size of the population of both MUL and TBC is often only a half of SC. These results evidences that both MUL and TBC achieve their objective in reducing GP code bloat.

With SSC and MSSC, the extent in which they reduce GP code bloat is often from 10% to 20% in comparison with SC. This table also shows that MSSC usually helps to reduce GP code bloat more than SSC. These results are consistent with the results in Chapter 7 where it has been shown that improving semantic locality leads to reducing code bloat and MSSC has the less bloat effect compared to SC and SSC.

Even if there is less bloat in some runs, it does not guarantee that the resulting solutions are smaller in size – best solutions could have an unrepresentative size relative to the rest of the population. Since the best solution in terms of fitness on the training sets (with VAL it is on the validating sets) is used as the final solution that will be tested for generalisation

Tab. 8.4: The average size of the best fitness.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	68.7	85.3	65.2	78.2	104.6	51.1	73.6	69.4
VAL	51.5	72.8	43.8	61.7	66.4	45.3	4.61	6.92
MUL20	34.1	51.3	26.0	44.4	41.6	25.5	50.5	45.7
MUL30	36.9	61.0	31.0	49.3	52.1	34.0	53.2	52.8
MUL40	50.1	69.8	34.2	44.4	64.0	47.7	58.9	57.4
TBC01	46.9	68.4	45.7	57.2	75.5	44.2	61.3	51.7
TBC02	32.4	49.1	32.8	42.5	59.9	25.9	48.6	41.7
TBC03	18.2	35.0	18.4	31.6	41.6	21.5	34.9	30.9
SSC12	63.1	88.4	68.4	75.0	103.8	50.6	77.2	68.1
SSC16	60.8	85.0	68.2	75.4	102.5	50.9	73.1	68.6
SSC20	62.1	85.0	72.2	70.1	103.4	49.4	70.6	65.8
MSSC12	54.1	77.7	67.7	66.8	102.1	50.0	69.3	65.0
MSSC16	62.8	75.3	64.8	66.0	102.1	44.2	66.6	57.6
MSSC20	57.8	76.9	68.4	68.1	101.2	50.8	61.7	57.9

ability, it is important to see the size of the final solutions of these methods. The results in Table 8.4 show that the reducing bloat of the above methods often helps them to find solutions with smaller size. The exceptions lie in some functions like F_2, F_3, F_8 with SSC and F_3 with MSSC. It should be noted here that while VAL, MUL and TBC often help to find solutions that are much smaller than SC, the reduction of the size of the solutions found by SSC and MSSC versus SC is not much. In some cases, F_3, F_5, F_6, they are only slightly smaller.

8.4.3 On the Ability of GP to Generalise

The previous subsections showed that improving locality like SSC and MSSC help to both improve GP performance and reduce GP code bloat, and that other methods (VAL, MUL, TBC) help to further reduce code bloat and find the solutions with better quality in terms of size, it is important to test the effect of these methods on GP's generalisation ability. To measure the generalisation ability of these methods, we tested the best individual found using the training set (with VAL it is the best individual in the validation set), for its

Tab. 8.5: The average of best fitness on testing set. The results of SSC and MSSC are printed bold face if they are significantly better than ones of SC. Note that the values are scaled by 10^2.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	2.53	4.48	0.83	1.93	16.2	19.5	38.3	14.8
VAL	3.08	4.08	1.19	1.86	18.2	21.8	38.1	14.4
MUL20	3.67	5.39	2.08	2.19	20.3	20.4	37.7	16.7
MUL30	3.17	4.26	2.29	2.38	19.6	19.5	36.0	18.0
MUL40	3.91	4.36	2.21	2.19	16.6	25.7	36.4	16.5
TBC01	2.13	4.26	1.03	1.70	22.0	18.1	37.7	15.7
TBC02	2.09	4.65	0.86	1.81	15.2	21.8	36.4	13.7
TBC03	4.72	5.25	1.45	2.23	15.2	18.2	38.8	18.1
SSC12	**1.26**	**2.15**	**0.56**	**1.04**	15.2	17.6	34.9	**12.5**
SSC16	**1.32**	**1.99**	**0.58**	**1.16**	16.4	16.3	34.9	14.2
SSC20	**1.52**	**1.96**	**0.56**	**1.19**	14.1	**13.4**	35.1	13.3
MSSC12	**1.48**	**2.14**	**0.55**	**0.91**	**12.7**	**13.7**	36.5	**11.4**
MSSC16	**1.73**	**1.88**	**0.50**	**1.33**	**12.7**	**11.1**	38.0	**12.6**
MSSC20	**1.41**	**2.12**	**0.46**	**1.26**	14.2	16.4	36.3	13.8

ability to generalise over independent test sets (See Table 8.1). We recorded the mean best fitness of the best individual on testing sets. The results are shown in Table 8.5.

It can be seen from this table that using a validation set method (VAL) or bloat control methods as MUL and TBC does not consistently help to improve the generalisation ability of GP even they help to reduce GP code bloat and find smaller solution. In fact, TBC is better than SC in some cases but the size of improvement is marginal and Wilcoxon signed-rank test shows it is not significant with a confident level of 95%. Generally, the average of the best fitness on the testing set of these methods is mostly equal to one of SC with some exception for MUL where the table shows that MUL is often much worse than SC on some functions (Functions F_1, F_3 and F_4). These results are consistent with the previous results in [42, 115, 52], it confirms that using validation set method or bloat control methods have a very limited impact on GP generalisation ablity.

Conversely, the table shows that both SSC and MSSC help to improve the generalisation ability of GP. We can see that the error of SSC and MSSC on the testing set are often

smaller than of SC. We also statistically tested the improvement of SSC and MSSC versus SC of the results in Table 8.5 using a Wilcoxon signed-rank test with a confidence level of 95%. In this table, if the improvement is statistically significant, the results are printed bold face. It can be seen that most of the improvement of SSC and MSSC is significant, with only exception on function F_7, where none of the improvement of SSC and MSSC is significant. The results of the statistical test also shows that MSSC more often significantly improve the generalisation ability then SSC.

Overall, the results in this section show that the advantage of semantic based crossovers (SSC and MSSC) over standard crossover is not only better on training data but also on unseen data. The results also confirm that using the validation set method or bloat control methods bring a limited impact on the generalisation ability of GP.

8.5 Conclusions

This chapter investigated the impact on the generlisation ability of GP on a number of methods. The experimental results showed that improving semantic locality of crossover often helps to improve GP's generalisation ability. The effect of using a validation set method and bloat control methods were also tested and the results showed that they both bring very little impact on the generalisation ability of GP, although they often help GP runs have faster runtimes and find more comprehensive soluitons. The next chapter will compare the role of semantic locality and syntactic locality of crossover.

Chapter 9

The Role of Semantic Locality and Syntactical Locality of Crossover

The previous chapters have shown that improving semantic locality of genetic operators leads to a significant improvement in GP performance. This raises a question whether locality in semantic or syntactic level are more important? This chapter investigates the role of syntactic locality and semantic locality of crossover in GP. First we propose a novel crossover for improving syntactic locality, *Syntactic Similarity based Crossover* (SySC). Then we compare this crossover with the crossover for improving semantic locality, Semantic Similarity based Crossover (SSC). The metrics analysed include GP performance, GP code bloat and the ability of GP to generalise. The results in this chapter have been published in [137].

9.1 Introduction

Locality is important in all search methods. In the field of Evolutionary Computation (EC), locality (continuity – small changes in genotype corresponding to small changes in phenotype) has long been seen as a desirable property of a representations [57, 69, 169, 171]. The preceding results have shown the benefits of improving semantic locality of crossover.

However, assuming a continuous genotype-phenotype mapping, one may then ask, whether it is better to design operators to control locality in genotype or phenotype space. On the side of the genotype space lies the advantage of simplicity: it is easy to measure and control locality directly in the space where the operators are applied. Thus virtually all previous work [57, 69, 169, 171] has relied on genotypic distance through syntactic metrics. On the other hand, at the cost of greater complexity, one might argue that phenotypic distances, being (presumably) more closely correlated with fitness, might lead to better metrics. Is this so? Is it worth the extra complication of designing semantic based control of operators? This chapter examines that question. We compare the semantic based control of crossover, SSC, with a new, syntactic based form.

9.2 Syntactical Similarity based Crossover

To create a crossover operator based on syntactic similarity, a structural distance/metric between any two trees is required. In this chapter, we use an extended version of tree distance that has been use by Ekart and Nemeth [41]. In other words, the syntactical distance between two trees is calculated as follows:

1. Make the two trees to be compared to have the same tree-structure (adding NULL nodes if necessary). Figure 9.1 gives an example of two trees which are completed by adding NULL nodes so that they have the same structure.

2. Count the distance between any two nodes located at the same position in the two trees. If two nodes are labeled with the same symbol, the distance between them is 0, otherwise the distance is 1.

3. Sum the distances computed in the previous step to form the distance of the two trees.

From this, a syntactic similarity relationship between two (sub)trees is defined in a similar way to the semantic similarity relationship. In other words, two subtrees are called

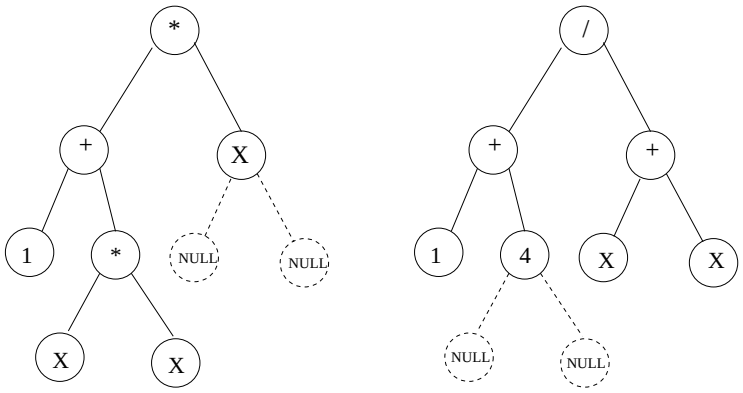

Fig. 9.1: Two trees are added the NULL nodes to have the same layout

syntactic similar if the syntactic distance (SyD) between them lies in an interval. Formally, two subtrees S_1 and S_2 are syntactic similar (SySi) if

$$\begin{aligned} SySi(S_1, S_2) = \quad & \textbf{if } \alpha < \mathrm{SyD}(S_1, S_2) < \beta \\ & \textbf{then } \text{true} \\ & \textbf{else } \text{false} \end{aligned}$$

where α and β are two predefined constants, the *lower* and *upper* bounds for syntactic sensitivity. In this chapter, α is set to 0, and β is set to 4. These values are calibrated from our experiments as good values for the performance of syntactic based crossover.

Based on SySi, a syntactic similarity based crossover is proposed. The crossover for improving syntactic locality, *Syntactic Similarity based crossover* (SySC), is inspired from the crossover for improving semantic locality, SSC. The algorithm for implementing SySC is similar to the one for performing SSC with the only difference being that syntactic similarity is checked instead of semantic similarity. Algorithm 6 shows how SSC works in detail. Several values of Max_Trial of SySC will be tested in the following subsections.

Algorithm 6: Syntactic Similarity based Crossover

```
select Parent 1 P_1;
select Parent 2 P_2;
Count=0;
while Count<Max_Trial do
    choose a random crossover point Subtree_1 in P_1;
    choose a random crossover point Subtree_2 in P_2;
    calculate the SySD between Subtree_1 and Subtree_2 on P
    if Subtree_1 is syntactically similar to Subtree_2 then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
choose a random crossover point Subtree_1 in P_1;
choose a random crossover point Subtree_2 in P_2;
execute crossover;
return true;
```

9.3 Semantic Locality Vs Syntactical Locality

This section compares the crossover for improving syntactic locality, SySC, with the crossover for improving semantic locality, *Semantic Similarity based Crossover* (SSC), and standard crossover (SC). The comparison is undertaken with three aspects of GP: GP performance, GP code bloat and the ability of GP to generalise. The tested problems are the eight problems that have been used in Chapter 8.

The basic GP parameter settings are given in Table 5.2. For SSC, three configurations similar to those in Chapter 8 are used. They are shorthanded as SSCX with X=12, 16 and 20. For SySC, three values of Max_Trial, 2, 4 and 6 were used. These values of Max_Trial guarantee from 60% (with Max_Trial=2) to nearly 100% (with Max_Trial =6) SySC successfully exchanges two syntactically similar subtrees. Totally, three configurations of SySC were tested that will be referred to as SySCX with X=2, 4, 6.

Tab. 9.1: The average of best fitness on training set. Note that the values are scaled by 10^2.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	1.54	3.42	0.49	1.40	2.45	1.34	11.1	13.3
SySC2	1.40	3.38	0.59	1.21	2.55	1.52	11.4	14.2
SySC4	1.73	3.22	0.82	1.25	2.45	1.86	11.9	14.9
SySC6	1.70	3.44	0.66	1.48	2.55	1.49	12.6	14.0
SSC12	0.85	1.85	0.28	0.62	2.11	0.92	8.59	9.89
SSC16	0.82	1.52	0.29	0.67	1.89	0.82	8.56	9.92
SSC20	0.86	1.53	0.30	0.71	1.87	0.74	8.63	9.24

9.3.1 Performance Comparison

To compare the performance of the three operators, we again recorded a classic performance metric, the mean best fitness. The results are shown in Table 9.1. It can be seen from this table that syntactically-bounded crossover (SySC) does not improve GP performance. The mean best fitness of SySC is mostly equal to SC. In some cases SySC is better than SC but in other cases SC is better than SySC. Conversely, the mean best that found by SSC is much better than both SC and SySC. This is consistent with the results in the previous chapters where it has been shown that SSC significantly improve GP performance.

We also statistically tested the performance of SSC versus SC and SySC using Wilcoxon signed-rank test with a confidence level of 95% (Table 9.1), confirming the significant improvement of SSC over both SC and SySC. Thus despite some advantages in implementation of SySC, SSC is a better bet than SySC in improving GP performance.

9.3.2 On the Code Bloat Effect

Although, improving syntactic locality of crossover does not improve the performance of GP, intuitively, it may help to reduce bloat compared to other crossovers. This subsection investigates the impact of SySC on the code bloat problem in GP. We compare it with the crossovers for promoting semantic locality. We measured the average size of individuals (number of nodes) over 50 generations, averaged over 100 runs and the average size of the

Tab. 9.2: The average of population size.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	52.4	60.5	46.1	55.8	70.5	54.5	50.6	48.1
SySC2	46.5	53.0	41.0	45.6	57.6	47.7	42.9	41.9
SySC4	42.5	48.1	38.1	43.1	53.6	44.8	38.3	38.0
SySC6	40.0	47.5	37.3	43.0	52.4	43.4	39.4	39.1
SSC12	47.3	57.9	43.9	49.7	69.3	45.2	46.7	42.2
SSC16	44.4	57.6	42.9	49.5	64.5	44.4	44.5	41.0
SSC20	46.4	56.7	42.8	49.2	64.9	42.8	43.6	38.1

Tab. 9.3: The average size of the best fitness.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	68.7	85.3	65.2	78.2	104.6	51.1	73.6	69.4
SySC2	59.8	73.2	57.4	56.9	80.9	46.4	60.7	59.5
SySC4	51.2	61.7	45.4	53.3	71.9	47.0	53.2	51.6
SySC6	54.2	63.0	43.5	53.9	68.8	42.0	56.6	54.4
SSC12	63.1	88.4	68.4	75.0	103.8	50.6	77.2	68.1
SSC16	60.8	85.0	68.2	75.4	102.5	50.9	73.1	68.6
SSC20	62.1	85.0	72.2	70.1	103.4	49.4	70.6	65.8

best fitness on the training set. The average size of individuals in the population is shown in Table 9.2 and the average size of the best fitness is depicted in Table 9.3.

Table 9.2 reveals that improving syntactic and semantic locality both helps to reduce code bloat, and in this respect, improving syntactic locality has a greater effect than improving semantic locality with some exceptions on function F_6 and F_8. We also investigated whether reducing the scale of change of SySC could further reduce GP code bloat, and indeed this was the case – but at the cost of poorer performance of SySC; the syntactic sensitivity used in this chapter is one of the best values found for the SySC's performance.

Table 9.3 shows that both promoting semantic and syntactical locality help crossover to find better solutions in terms of the size of the best fitness. It can be seen from the table that the average size of the best fitness on the training set of SySC are always smaller than ones of SC. The results for SSC are closer to SC than SySC and, in this respect, improving syntactical locality also has a greater effect than improving semantic locality. The average

Tab. 9.4: The average of best fitness on testing set. The results of SSC are printed bold face if they are significantly better than ones of SC.

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SC	2.53	4.48	0.83	1.93	16.2	19.5	38.3	14.8
SySC2	2.98	4.17	0.99	1.74	14.7	20.0	37.1	16.7
SySC4	2.94	3.99	1.18	1.68	14.3	19.1	37.1	17.2
SySC6	2.84	4.40	1.02	1.92	14.6	19.2	36.2	18.1
SSC12	**1.26**	**2.15**	**0.56**	**1.04**	15.2	17.6	34.9	**12.5**
SSC16	**1.32**	**1.99**	**0.58**	**1.16**	16.4	16.3	34.9	14.2
SSC20	**1.52**	**1.96**	**0.56**	**1.19**	14.1	**13.4**	35.1	13.3

size of the best fitness that found by SySC is always smaller than one of SSC. This can be seen as the result of reducing further bloat of SySC in comparison with SSC.

9.3.3 On the Ability to Generalise

The previous subsection has shown that improving syntactical locality helps to maintain the performance of GP (the performance of GP with SySC is mostly equal to SC), while remarkably reduce GP code bloat. Especially, SySC alway finds solutions (the best fitness individuals) that are much simpler (smaller) than SC. Therefore, it is important to test if this kind of reducing code bloat can help to improve the generalisation ability of GP.

To measure the generalisation ability, we again tested the best individual found on the training set, for its ability to generalise over independent test sets (see Table 8.1). These values are then averaged over 100 runs and the results are shown in Table 9.4.

It can be seen from this table that although help to reduce GP code bloat and find simpler solutions, improving syntactical locality of a crossover does not improve the generalisation ability of GP. The table shows that the average fitness on the testing set of SySC and SC are mostly equal. On some functions (F_2, F_4, F_5, F_8), it seems that SySC slightly improve the ability of GP to generalise. However, the size of improvement is small and not significant (using Wilconxon signed-rank test with a confidence level of 95%). On other function, SySC and SC are most equal and sometimes SySC is worse (F_1, F_8). By contrast,

improving semantic locality often leads to significantly improve the generalisation ability of GP. Generally, the results in this section confirm the more important role of semantic locality of crossover in comparison with syntactical locality.

9.4 Conclusions

This chapter investigates the role of semantic locality and syntactic locality of crossover. We propose a novel syntactic similarity based crossover for improving syntactic locality. We compare syntactic similarity based crossover with the crossover for improving semantic locality, *Semantic Similarity based Crossover*, and with standard crossover on a number of aspects of GP. The experimental results show that while improving semantic locality helps to significantly improve GP performance, reduce GP code bloat and substantially enhance the ability of GP to generalise, improving syntactic locality only leads to reducing code bloat and slightly improving the ability of GP to generalise over standard crossover. The results confirm the more important role of semantic locality than syntactic locality. These results could be seen as an evidence to attract GP researchers to pay more attention to semantics when designing their algorithms. The next chapter will examines the role of semantics for problem difficulty and their effect on fitness landscape.

Chapter 10

A Study of Problem Difficulty and Fitness Landscape

The previous chapters have shown the advantages of semantic based crossovers in solving a class of problems. However, how these crossovers deal with increasingly difficult problems and how they affect search on the fitness landscape are still open questions. This chapter presents a study of search fitness landscape of problems with increasing level of difficulty. First, an investigation of semantics based crossovers with increasingly difficult problems is presented. The question it aims to address is how semantic based crossovers deal with increasingly difficult problems. Next, the chapter focuses on examining the fitness landscape with the aim to answer the question if improving semantic locality of an operator helps to smooth out the fitness landscape of a problem. Some results in this chapter have been published in [139].

10.1 Semantic based Crossovers with Difficulty Increased Problems

This section studies the behaviour of semantic based crossovers when the difficulty of a problem is increased. A brief review of previous work on problem difficulty is given first.

The experimental settings, results and discussion are presented after that.

10.1.1 Related Work

There have been a number of studies addressing problem difficulty for Genetic Programming (GP). In the early days of GP, Koza introduced semi-empirical formula for estimating the number of trials needed to solve a problem with a specified success probability [98]. Since then, three have been two main strands of research on GP problem difficulty. The first strand is to build a test suite of tunably difficult problems and the second strand is to quantify the difficulty of a problem.

In the first strand, Koza [98] introduced the set of tunably difficult problems including the problems for learning Boolean multiplexers and the Boolean parity functions. In his second book in the series on GP [99], Koza provided polynomials (a sextic and a quintic), Boolean symmetry (5- and 6-symmetry), and Fourier sine series (3- and 4-terms). Gathercole and Ross [53] proposed the MAX test suite, while Punch et al. [160] introduced a tunably difficult royal tree problem. O'Reilly and Goldberg [56, 149] have also proposed ORDER and MAJORITY that have been claimed to be the GP version of the onesmax problem in GA. In a thoughtful paper, Daida et al. [34] examined a tunable problem, the binomial-3 function with varying ephemeral random constant (ERC) ranges. In this case, GP problem difficulty increases with the increased range of ERC values when ERC is greater than 1. The authors argued that the conflict between content and context is largely responsible for increased difficulty of these problems. In a recent work, Hao et al. [70] proposed ORDERTREE, a natural analogue of the onemax problem, in which the difficulty of the problem can be tuned by increasing problem size or by increasing the non-linearity in the fitness structure.

In the second strand, O'Reilly [148] used the metaphor of fitness landscape to study problem difficulty. This work is a GP extension of the work on fitness landscape analysis in GA [118, 72]. The general knowledge is that the more rugged the fitness landscape, the more difficult the problem is. Kinnear [95] used landscape autocorrelation to study

the difficulty of problems. The results showed that there is a strong correlation between adaptive walks and the difficulty of problems. Approaching in a different way, Gustafson et al. [61] studied the relationship between code growth and problem difficulty. The authors showed that there is a correlation between code bloat and problem difficulty in which increased problem hardness induces higher selection pressure and less genetic diversity, which both contribute toward an increased rate of code growth. In a series of work [183, 181, 185], Vanneschi et al. proposed Fitness Distance Correlation (FDC) and Negative Slope Coefficient (NSC) to measure problem difficulty. These methods have been tested on a number of problems and the results show that they are useful in qualifying the difficulty of problems. However, the weakness of FDC is that it needs to know the optimal solutions beforehand, while NSC can not compare the difficulty of different problems.

This section uses a class of increasingly difficult problems, the binomial-3 [34], as a tool to investigate the performance of semantic based crossovers. The previous chapters have shown that semantic based crossover help to significantly improve the performance of GP. Therefore, it is interesting to see how these crossovers behave when the difficulty of problems is increased. In other words, it is important to investigate if these crossovers help to improve GP performance when the problems become harder. This section aims to address the above questions.

10.1.2 Experimental Settings

In this section we use a class of symbolic regression problems that are scaled in difficulty to observe the behaviour of semantic based crossovers. This class is the binomial-3 problem [34]. The binomial-3 problem consists of approximating the function $f(x) = (x + 1)^3$. Using the terminals set of {x, ERC}, it has been shown in [34, 61] that the difficulty of this problem is increased with the changing of ERC. In other words, if ERC ≥ 1, the difficulty of binomial-3 is increased by increasing ERC. In this experiment we use three ranges of ERC which has been use in [34, 61], they are [-1, 1], [-10, 10] and [-100, 100]. Binomial-3 with these ranges of ERC will be referred to as Bin1, Bin10 and Bin100, respectively.

The basic experiment settings are as in Table 5.2 where the only difference is the terminal set. The terminal set now is {x, ERC} with ERC varies in three ranges for binomial-3. The training set for all problems include 20 equidistant over interval (0, 1].

We examined the behaviour of semantic based crossovers (SSC and MSSC) when they deal with increasingly difficult problems and compared them with standard crossover (SC). The lower bound of semantic sensitivity for SSC was set at 10^{-3} and the upper bound one was set to 0.4. There values of Max_Trial, 12, 16, 20 were tested. They are the values that have been used in the experiments in the previous chapters. These three configurations of SSC will be denoted as SSCX with X=12, 16, 20. Similarly, the lower bound semantic sensitivity for MSSC was set to 10^{-3}. Also, there values of Max_Trial, 12, 16, 20 for MSSC were tested. These three configurations of MSSC will be referred to as MSSCX with X=12, 16, 20. For all configurations, 100 runs were performed.

10.1.3 Results and Discussion

Since there is not any method that can exactly quantify the difficulty of different problems, here we use three metrics that are often used to characterise problem difficulty. These three metrics are: the number of solutions found by GP, the rate of minimizing the best fitness, and code growth. The number of solutions is measured by the number of successful runs (a run is considered successfully if it hits on all fitness cases). The rate of minimizing the best fitness is measured by the best fitness of a run and code growth is quantified by the average of individual size in the population. We recored these values, averaged over 100 runs and the results are shown in Table 10.1. In this table, No. Solutions is the number of solutions found by each crossovers, Best Fitness is the best fitness of each run and averages over 100 runs and Pop. Size is the size of individuals in the population averaged over all generations and 100 runs.

Table 10.1 shows that the difficulty of binomial-3 is increased by the increase of ERC from Bin1 to Bin100. It is reflected by both the number of solutions found for each problem and the rate of minimising the best fitness. It can be seen that the number of solutions are

Tab. 10.1: The Comparison of Crossovers on Binomial-3.

Crossovers	No. Solutions			Best Fitness			Pop. Size		
	Bin1	Bin10	Bin100	Bin1	Bin10	Bin100	Bin1	Bin10	Bin100
SC	8	2	0	2.18	3.81	5.68	59.8	60.1	62.7
SSC12	19	14	10	1.03	2.14	2.47	51.8	51.9	52.5
SSC16	19	13	6	0.91	1.92	2.41	49.0	50.6	51.0
SSC20	15	12	9	1.05	2.19	2.71	51.0	49.3	49.8
MSSC12	35	26	19	0.85	1.66	1.94	48.2	49.2	44.7
MSSC16	29	19	13	0.99	1.48	1.89	46.1	43.4	45.1
MSSC20	31	23	17	1.02	1.71	1.81	45.3	46.3	43.7

reduced from Bin1 to Bin100 while the mean of the best fitness rises from Bin1 to Bin100. These results are consistent with the previous results [34] where the authors also showed that the difficulty of Binomial-3 is increased by increasing the value of ERC with ERC greater than 1.

While the results of both number of solutions and the mean best fitness provide evidence for the increasing difficulty of binomial-3 from Bin1 to Bin100, the results on code bloat seems to be a different story. The table obviously shows that there is not any relationship between code growth and problem difficulty at least with this GP parameter settings. It seems like the relationship between code growth and problem difficulty investigated in [61] is also dependent on GP settings.

Comparing between crossovers, the table clearly shows that semantic based crossovers help to improve the performance of GP even when problems become harder. It is shown by the fact that the number of solutions found by SSC and MSSC are always greater than the ones found by SC, and the mean best fitness of SSC and MSSC are consistently smaller than those of SC. The statistical test (using Wilcoxon signed-rank test) shows that all the improvements of SSC and MSSC versus SC are significant with a confident level of 95%. It also seems that when problems become harder, the size of the improvement of SSC and MSSC are even bigger. The table also shows that MSSC is often better than SSC and both SSC and MSSC help to reduce code bloat while MSSC help to reduce it to a further extent.

In summary, on the tunably difficult binomial-3 problem we see a performance advantage for the semantic based operators over standard crossover. Therefore, paying attention to semantics can effectively reduce difficulty in this case.

10.2 Examining Fitness Landscape of Semantic based Crossovers

The earlier chapters have shown that semantic based operators help to significantly improve GP performance. The improvement of semantic locality leads to the enhancement of a number of aspects of GP systems. In this section, we investigate the impact of these semantic based operators on the fitness landscape. The main aim of this section is to examine if improving semantic locality of an operator can help to smooth out the fitness landscape.

The previous section in this chapter showed that we can scale the difficulty of the binomial-3 problem by changing ERCs, and that semantic based operators can deal well with increasingly difficult problems. Therefore, it is also informative to see if the difficulty of these problems is caused by a change to the fitness landscape or by other factors. The section is organised as follows. We first discuss the related work on fitness landscape in Evolutionary Computation. Two methods used to qualify fitness landscape are briefly presented after that. Next are experimental settings, results and discussion.

10.2.1 Related Work

A fitness landscape is a way of describing the search space of a problem in evolutionary algorithms. The concept of fitness landscape was first proposed by Wright [199] to study the evolutionary process in biology. Since then, it has widely been used to model the problem difficulties in evolutionary algorithms (EAs) [163, 36]. A Fitness landscape uses metaphors from nature such as peaks, hills, valleys, ridges, basins, watersheds etc. to characterise the search space of a problem that EAs might encounter. A fitness landscape

with many local peaks surrounded by deep valleys is called rugged. For the problems with this fitness landscape, it is more difficult to find solutions (the highest peaks), since the algorithms can be trapped in any local peak. Generally, the more rugged the fitness landscape, the more difficult the problem is. If all genotypes have the same fitness values, on the other hand, a fitness landscape is said to be flat. For this fitness landscape, an algorithm can not exploit knowledge (e.g., fitness gradients) from the search space.

In practice, the visualisation of the whole search space of a problem is problematic. Therefore, a number of methods that attempt to describe the structure of fitness landscapes have been proposed [190, 163, 188]. In fact, before describing a fitness landscape, its primary components must be defined [87]. The first component is the representation, i.e., how we encode the problem on a genotype structure to represent all potential solutions of the problem. The second component is the operators that transform a candidate solution from one point to another point in the search space. The third component is comprised of a function (the fitness function) which allows us to assign a measure of quality (fitness) to each candidate solution, a fitness space, and a partial order of solutions over the fitness space.

The structure of fitness landscapes influences the ability of an evolutionary algorithm to perform an efficient search. There are several characteristics associated with the landscape that define its structure. These characteristics include number, type, magnitude, and the sizes of the optima, and of their basins of attraction. Researchers have investigated the different aspects of the structure of fitness landscapes, such as landscape deceptiveness [54, 55], modality [3] and ruggedness [91, 190], to understand the nature of evolutionary search under different conditions.

Some authors suggest the use of an operator that minimises distance traveled in the search space when studying fitness landscape [67, 181, 184]. In this research, we follow the work in [95, 87, 88, 187] to define the metric in terms of the operators. It is reasonable since the main objective of this research is to compare the characteristics of the fitness landscape with different operators.

10.2.2 Techniques Used

To characterise the fitness landscape of these methods, we use two well known techniques. The first technique is the autocorrelation function and the second method is the information content. They are detailed below:

Correlation Analysis: correlation analysis is a set of techniques for charactering problem difficulty by measuring the correlation between the fitness of neighbouring points [36]. There are three main techniques of correlation analysis of these autocorrelation metric of fitness landscape will be used in this study.

Using autocorrelation function to study fitness landscapes was first proposed by Weinberger [190]. For a given fitness landscape with f (the fitness function), a starting point s_0 is randomly selected. A mutation operator is used to create a neighbouring point s_1 of s_0. Repeat this proccess N time to get a random walk of N steps $F = \{f(s_i)\}_{i=0}^{N}$. Then the autocorrelation function of this random walk is defined as follows:

$$\rho(h) = \frac{R(h)}{s_f^2} \tag{10.1}$$

where h is the distance between two points in the random walk. s_f^2 is variance of the sequence and calculated as follows:

$$s_f^2 = \frac{\sum_{i=0}^{N}(f(s_i) - m_F)^2}{N+1} \tag{10.2}$$

and $R(h)$ is autocovariace funtion of sequence F. For each h, R(h) is estimated by:

$$R(h) = \frac{\sum_{i=0}^{N-h}(f(s_i) - m_F).(f(s_{i+h}) - m_F)}{N-h+1} \tag{10.3}$$

where m_F is mean of fitness sequence F, $m_F = \frac{1}{N+1}\sum_{i=0}^{N}(f(s_i))$. The autocorrelation function indicates the correlation between points that are separated by a distance h. A smooth landscape is highly correlated as the fitness difference between a point with its neighouring is small. In this case the autocorrelation function is greater. Conversely, if a fitness landscape is rugged, the fitness difference a point with its neighouring is high, the

autocorrelation function is smaller.

Information Content: a method for charactering a fitness landscape based on the concept of information content was first proposed by Vassilev et al. [188]. Similar to an autocorrelation function, a random walk of N steps, $F = \{f(s_i)\}_{i=0}^{N}$ is obtained, first. A sequence is driven from F, $S(\epsilon) = s_1, s_2, ..., s_N$, to present this random walk for each ϵ, where

$$s_i = \Psi(i, \epsilon) \tag{10.4}$$

and

$$\Psi(i,\epsilon) = \begin{cases} -1 & \text{if } f_i - f_{i-1} < -\epsilon; \\ 0 & \text{if } |f_i - f_{i-1}| \leq \epsilon; \\ 1 & \text{if } f_i - f_{i-1} > \epsilon; \end{cases} \tag{10.5}$$

The parameter ϵ is a real-number from the range [0..L], where L is the greatest value in the sequence F. ϵ determines the accuracy of calculation of $S(\epsilon)$. If ϵ=0, function $\Psi(i, \epsilon) =$ will be very sensitive, and insensitive when ϵ=L.

After that, four measures of entropy and amount of fitness change during the random walk are calculated as follows:

1. *Information Content* ($H(\epsilon)$): indicates the ruggedness of a landscape
2. *Partial information content* ($M(\epsilon)$): indicates the modality of a landscape.
3. *Information stability* ($\epsilon*$): indicates the magnitude of optima in a landscape.
4. *Density-basin information* ($h(\epsilon)$): characterises the structure of a landscape around optima.

10.2. EXAMINING FITNESS LANDSCAPE OF SEMANTIC BASED CROSSOVERS

The information content characterises the ruggedness of a fitness landscape. It is defined as follows:

$$H(\epsilon) = \sum_{p \neq q} P_{[pq]} log_6 P_{[pq]} \tag{10.6}$$

where the probability $P_{[pq]}$ are the frequencies of 6 posible block qp, $p \neq q$, of elements from $S(\epsilon)$. They are defined as

$$P_{[pq]} = \frac{N_{[pq]}}{N} \tag{10.7}$$

where $N_{[pq]}$ is the number of occurrences of pq in $S(\epsilon)$.

The partial information content is defined as

$$M(\epsilon) = \frac{\mu}{N} \tag{10.8}$$

where N is the length of frequency $S(\epsilon)$. μ is calculated by calling a recursive function of three integer arguments and called by $\Phi_S(1,0,0)$, with $\Phi_S(i,j,k)$ is defined as follows:

$$\Phi_S(i,j,k) = \begin{cases} k & \text{if i>N} \\ \Phi_S(i+1,i,k+1) & \text{if } i = 0 \text{ and } s_i \neq 0; \\ \Phi_S(i+1,i,k+1) & \text{if } j > 0 \text{ and } s_i \neq 0 \text{ and } s_i \neq s_j; \\ \Phi_S(i+1,j,k) & otherwise; \end{cases} \tag{10.9}$$

When $M(\epsilon)$=0 it is an indication that there is no slope in the random walk and when $M(\epsilon)$=1, it is an indication that there is a maximal of multi-modality in the random walk. Moreover, for a given partial information content, $M(\epsilon)$, the number of optimal (Optimal No. in Table 10.3, Table 10.4, and Table 10.5) in the random walk can be calculated as $\lfloor \frac{N.M(\epsilon)}{2} \rfloor$.

The last measurement,density-basin information, $h(\epsilon)$, can be calculated as

$$h(\epsilon) = \sum_{p \subseteq \{-1,0,1\}} P_{[pp]} log_3 P_{[pp]} \tag{10.10}$$

where pp is one of three sub-blocks, 00, -1-1, 11. The high value of $h(\epsilon)$ presents the high number of peaks in a small area of the landscape. The low value means that the optima is isolated.

In general, higher values of information content, partial information content, number of optimas in the landscape, density-basin information define a more rugged landscape which is more difficult to search. A smaller ones define a smoother landscape which is easier to a search.

10.2.3 Experimental Settings

We use the class of problems investigated earlier in this chapter (binomial-3) to determine the impact of semantic based operators on their fitness landscape. There are two main objectives of this experiment. The first objective is to determine whether increasing problem difficulty might stem partly from changing the fitness landscape, or from other factors. The second objective is to test if improving semantic locality of an operator helps to smooth out the fitness landscape of a problem.

To experimentally investigate the fitness landscape of the above problems, a walk of N steps need to be created first. Most of the previous work studied the fitness landscape of a mutation [67, 181, 184, 95, 87] where a random walk of N steps can easily be generated by applying the mutation N times. Since this research mainly focuses on studying the fitness landscape of crossovers, we followed Riley and Ciesielski [164] in creating a random walk of N steps as follows:

- An individual i_0 is randomly selected from the search space.
- set $s = 1$

- Repeat
  - Another individual i_s, $i_s \neq i_{s-1}$, is randomly selected from the search space.
  - Crossover is performed between i_s and i_{s-1} with probability $P_{crossover}$ resulting in two new individuals i'_1 and i'_2, both are neighbours of (a single step from) i_{s-1}. Set i_s=i'_1 and discard i'_2.
  - i_s undergoes mutation with probability $P_{mutation}$
  - set $s = s + 1$
- Until $s > N$

In this experiment, we set $P_{crossover}$=0.9 and $P_{mutation}$=0.05 as the same with the previous experiments. We compare the fitness landscape of Standard Crossover (SC), with Syntactic Similarity based Crossover (SySC) and two semantic based crossover, Semantic Similarity based Crossover (SSC) and the Most Semantic Similarity based Crossover (MSSC). For each crossover, a random walk of 10000 steps was generated. All fitness values of individuals encountered during the random walk were recorded. 500 random walks were conducted for each problem, making making a total of 5,000,000 fitness evaluations for each experiment.

The lower semantic sensitivity used for SSM is 10^{-3} and the upper semantic sensitivity is set to 0.4. The Max_Trial of SSM is set at 12. These values of the parameters for SSM have been shown are good values for the performance of SSM in the previous chapters. For MSSC we set Max_Trial=12 and for SySC Max_Trial is set to 4. Similarly, they are the values for the good performance of these crossovers.

10.2.4 Results and Discussion

To characterise the fitness landscape of these crossovers, the autocorrelation and information content of SC, SySC, SSC and MSSC were measured. The results of autocorrelation

Tab. 10.2: Autocorrelation Analysis (larger values are better).

Distance (h)	Crossovers	Bin1	Bin10	Bin100
1	SC	0.714	0.712	0.704
	SySC	0.719	0.715	0.712
	SSC	0.776	0.758	0.744
	MSSC	0.782	0.776	0.764
2	SC	0.633	0.637	0.635
	SySC	0.644	0.648	0.650
	SSC	0.727	0.706	0.695
	MSSC	0.735	0.734	0.727
4	SC	0.521	0.535	0.538
	SySC	0.544	0.556	0.566
	SSC	0.655	0.627	0.621
	MSSC	0.665	0.670	0.672
8	SC	0.392	0.413	0.419
	SySC	0.421	0.445	0.461
	SSC	0.556	0.522	0.519
	MSSC	0.571	0.582	0.593
16	SC	0.265	0.287	0.294
	SySC	0.316	0.330	0.324
	SSC	0.438	0.397	0.396
	MSSC	0.453	0.471	0.493
32	SC	0.162	0.179	0.182
	SySC	0.219	0.213	0.227
	SSC	0.318	0.267	0.269
	MSSC	0.328	0.347	0.375

analysis were shown in Table 10.2. Table 10.3, Table 10.4 and Table 10.5 show the results of information content analysis for Bin1, Bin10 and Bin100, respectively.

Table 10.2 shows that improving syntactic locality of crossover has a limited impact on the fitness landscape. It can be seen that the value of the autocorrelation function of SySC is only slightly greater than of SC. The statistical test using a Wilcoxon signed-rank test with a confident level of 95% shows that it is mixed, meaning that only some of the values of SySC is significantly greater than SC. These results explain why SySC does not help to improve the performance of GP in comparison with SC.

Conversely, improving semantic locality of crossover helps to smooth out the fitness

Tab. 10.3: Information Content Analysis of Bin1 (smaller values are better).

Epsilon(ϵ)	Crossovers	H(ϵ)	h(ϵ)	M(ϵ)	Optima No.
0	SC	0.733	0.629	0.475	2376
	SySC	0.688	0.629	0.495	2479
	SSC	0.686	0.626	0.472	2374
	MSSC	0.686	0.627	0.465	2371
0.2	SC	0.683	0.572	0.257	1284
	SySC	0.671	0.569	0.257	1280
	SSC	0.568	0.487	0.186	931
	MSSC	0.545	0.461	0.172	859
0.4	SC	0.572	0.476	0.195	979
	SySC	0.556	0.468	0.193	964
	SSC	0.437	0.380	0.135	677
	MSSC	0.420	0.361	0.126	631
0.6	SC	0.497	0.417	0.164	817
	SySC	0.479	0.408	0.160	801
	SSC	0.373	0.335	0.115	576
	MSSC	0.358	0.313	0.107	537
0.8	SC	0.443	0.378	0.144	717
	SySC	0.426	0.369	0.141	702
	SSC	0.335	0.305	0.104	522
	MSSC	0.322	0.291	0.097	488
1	SC	0.402	0.351	0.130	651
	SySC	0.387	0.343	0.127	635
	SSC	0.310	0.288	0.096	487
	MSSC	0.298	0.274	0.091	456

landscape of a problem to a greater extent. Obviously, the value of autocorrelation function of SSC and MSSC are always much greater than those of SC and SySC. It means that the fitness value of the individuals in the population of SSC and MSSC has stronger correlation than SC and SySC and hence the fitness landscape of the population that are constructed by SSC and MSSC are smoother than one of both SC and SySC. We also statistically tested the difference between autocorrelation values of SSC and MSSC with SC and SySC using a Wilcoxon signed-rank test and the results of this test shows all the differences are significant with a confidence level of 95%. The table also shows that the value of the autocorrelation function of MSSC is alway greater than SSC. This explains why MSSC

Tab. 10.4: Information Content Analysis of Bin10 (smaller values are better).

Epsilon(ϵ)	Crossovers	H(ϵ)	h(ϵ)	M(ϵ)	Optima No.
0	SC	0.734	0.657	0.474	2376
	SySC	0.698	0.647	0.492	2462
	SSC	0.706	0.652	0.470	2353
	MSSC	0.693	0.641	0.473	2366
0.2	SC	0.671	0.579	0.265	1325
	SySC	0.664	0.578	0.265	1329
	SSC	0.604	0.527	0.216	1081
	MSSC	0.585	0.510	0.205	1025
0.4	SC	0.577	0.502	0.214	1072
	SySC	0.569	0.497	0.213	1065
	SSC	0.498	0.441	0.170	852
	MSSC	0.473	0.421	0.158	794
0.6	SC	0.518	0.457	0.188	942
	SySC	0.509	0.453	0.186	931
	SSC	0.442	0.400	0.150	754
	MSSC	0.413	0.378	0.139	696
0.8	SC	0.477	0.428	0.172	862
	SySC	0.467	0.420	0.169	847
	SSC	0.406	0.375	0.139	695
	MSSC	0.377	0.352	0.128	642
1	SC	0.444	0.407	0.161	805
	SySC	0.434	0.398	0.158	788
	SSC	0.380	0.358	0.131	656
	MSSC	0.352	0.338	0.121	607

produces better performance than SSC as has been shown in the previous chapters.

In comparison between the problems, the table shows very little difference between Bin1, Bin10 and Bin100. In fact, it can be seen from the table that in some case the values of autocorrelation function of B100 and Bin10 are greater than Bin1 and in some other case they are converse. Therefore, it is difficult to conclude that the difficulty of Bin10 and Bin100 are stemed from their fitness landscape. We believe that the increasing difficulty of Bin10 and Bin100 versus Bin1 is because of the conflict between context and content as they have been shown by Daida et al. [34].

Table 10.3, Table 10.4 and Table 10.5 are consistent with Table 10.2 in which they con-

Tab. 10.5: Information Content Analysis of Bin100 (smaller values are better).

Epsilon(ϵ)	Crossovers	H(ϵ)	h(ϵ)	M(ϵ)	Optima No.
0	SC	0.741	0.717	0.445	2226
	SySC	0.725	0.706	0.456	2281
	SSC	0.724	0.711	0.440	2220
	MSSC	0.712	0.716	0.443	2217
0.2	SC	0.639	0.570	0.266	1327
	SySC	0.625	0.565	0.262	1310
	SSC	0.585	0.529	0.232	1160
	MSSC	0.558	0.517	0.223	1116
0.4	SC	0.559	0.507	0.226	1131
	SySC	0.546	0.499	0.221	1108
	SSC	0.496	0.464	0.193	979
	MSSC	0.471	0.447	0.185	927
0.6	SC	0.513	0.473	0.207	1035
	SySC	0.496	0.465	0.203	1013
	SSC	0.451	0.432	0.181	905
	MSSC	0.421	0.415	0.168	849
0.8	SC	0.479	0.453	0.195	927
	SySC	0.463	0.444	0.191	952
	SSC	0.423	0.418	0.172	861
	MSSC	0.392	0.398	0.161	806
1	SC	0.450	0.436	0.185	926
	SySC	0.446	0.428	0.183	907
	SSC	0.400	0.406	0.162	830
	MSSC	0.370	0.386	0.155	778

firm the limited influence of improving syntactic locality of crossover on a fitness landscape. It can be observed that the value of information content, partial information content, information stability, and density-basin information of SySC are only marginally smaller than that of SC and sometimes these values even greater than the those of SC. The results of the statistical test show that only a few of the differences between SySC and SC in these tables is significant with a confidence of 95%.

On the other hand, these tables show the ability to smooth out the fitness landscape by improving semantic locality. Definitely, the values that characterise for information content of SSC and MSSC are always smaller than both SC and SySC, meaning that SSC

and MSSC help not only to reduce the ruggedness of the fitness landscape but also to decrease the number of local optima in the landscape and decrease the magnitude of this local optimal. We also statistically tested the difference between SSC and SSM with SC and SySC in Table 10.3, Table 10.4 and Table 10.5 using Wilcoxon signed-rank test and the results of this test show most of the difference is significant with a confidence level of 95% with only some exceptions when $\epsilon = 0$.

Comparing between SSC and MSSC, these tables again show that the fitness landscape constructed by MSSC is smoother than SSC. It is reflected by the fact that the values for information analysis of MSSC is consistently smaller than SSC. Investigating the information analysis of the three problems, Bin1, Bin10 and Bin100, it is hard to draw any conclusions. Therefore, the difficulty of these problems are not caused by changing the characteristic of their fitness landscape.

10.3 Conclusion

This chapter investigated problem difficulty and the fitness landscape in GP with semantic based operators. The first section examined the behavious of semantic based crossovers with increasingly difficult problems. In this section, a class of tunably difficult problems, binomial-3, were used in the experiments. Semantic based crossovers were tested on these problems and the results showed that semantic based crossovers (SSM and MSSC) can deal well with increasingly difficult problems. In other words, they show that semantic based crossovers still help to significantly improve the performance of GP in comparison with standard crossover.

The next section studied the fitness landscape of semantic based crossovers. We analysed the fitness landscape of the above problems using two well know techniques, an autocorrelation function and information content. The results showed that semantic based crossovers help to smooth out the fitness landscape of these problems. These results again have shed some light on the performance of semantic based operators. The next chapter will examine the role of semantic based crossover in solving Boolean problems.

Chapter 11

Semantic based Crossovers for Boolean Problems

This chapter investigates the role of semantic diversity and locality of crossovers in Genetic Programming (GP) for Boolean problems. We design several new crossovers based on subtree semantics. They can be categorised into two classes depending on their purposes: promoting semantic diversity or improving semantic locality. We test the operators on several well-known Boolean problems, comparing them with Standard crossovers and with the Semantic Driven Crossover (SDC) of Beadle and Johnson [11]. The experimental results show the positive effects both of promoting semantic diversity, and of improving semantic locality, in crossover operators. They also show that the latter has a greater positive effect on GP performance than the former. Some important aspects of GP with these crossovers are also investigated that provide evidence for the improvement of semantic based crossovers. Some preliminary results in this chapter have been published in [136].

11.1 Introduction

The results in the previous chapters have shown that promoting semantic diversity is useful. However, improving semantic locality is even more important and it often leads

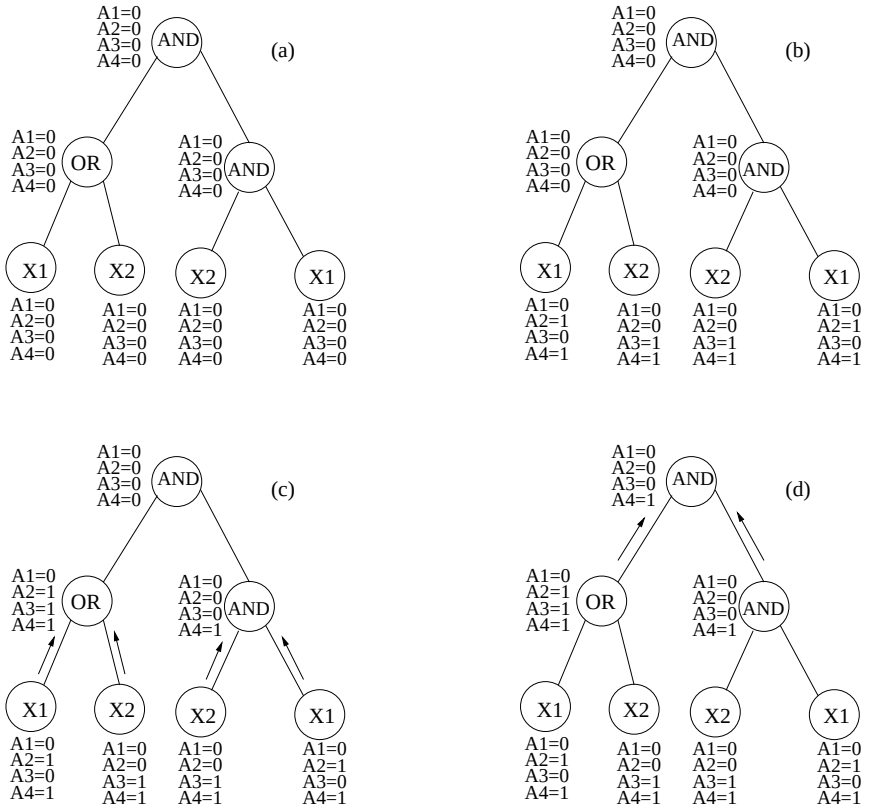

Fig. 11.1: An individual and the process of evaluating the value of its attributes. This illustrates the initialisation (a) and the evaluation of its attributes (b, c, d) by propagation up toward the root note.

to a further improvement of GP performance in solving real valued symbolic regression problems. The previous chapters also demonstrate that limiting semantic change between two subtrees in crossover to an intermediate range leads to improve both diversity and locality of that crossover on real valued problems. Potentially, though, discrete domains could show a different behaviour: the benefits of an intermediate semantic change might disappear with a quantised fitness function. In other words, the questions addressed here are whether semantic diversity and semantic locality are important for Boolean problems and if the crossovers in the early chapters (SAC and SSC) work well on Boolean problems. This chapter aims to answer the above questions.

11.2 Measuring Semanatics

The way to measure semantics for Boolean problems in this chapter is similar to the approach in Chapter 6. In other words, semantics is measured based on fitness cases of the problems. Again, to speed up the process of semantic checking, semantics of a subtree is stored in the root node of that subtree. In Figure 11.1, four attributes (A_1, A_2, A_3, A_4) are used to represent the semantics of a problem with only two input variables. These attributes are evaluated bottom-up, as shown in figure 11.1, in which the values are initialised to zero (subfigure a), then propagated upwards (subfigures b, c, d).

Based on *Subtree Semantics*, a *Subtree Semantics Distance* between two subtrees is defined in a similar way to *Sampling Semantics Distance* (SSD) in Chapter 4. From that, two semantic relationships between subtrees, *Semantic Equivalence* (SE) and *Semantic Similarity* are defined in the same way with their definitions in Chapter 4.

11.3 Semantic based Crossovers

This section presents some semantic based crossovers for Boolean problems. Two crossovers for promoting semantic diversity are presented first. After that, two crossovers for improving semantic locality are detailed.

11.3.1 Crossover for Promoting Semantic Diversity

The first crossover that promotes semantic diversity in Boolean problems is Semantic Aware Crossover (SAC). As has been presented in the previous chapters, SAC works by aborting crossovers that exchange semantically equivalent subtrees. However, for Boolean problems, SAC cannot guarantee to produce children semantically different from their parents. For example, consider figure 11.2(a), showing part of a GP tree with attached semantic attributes (X, Y and Z represent whole subtrees, not merely single nodes). Assume that X is the point selected for crossover. Suppose that it is replaced by subtree $X1$, with the semantics shown in 11.2(b). Although the semantics of X and $X1$ differ, the semantics

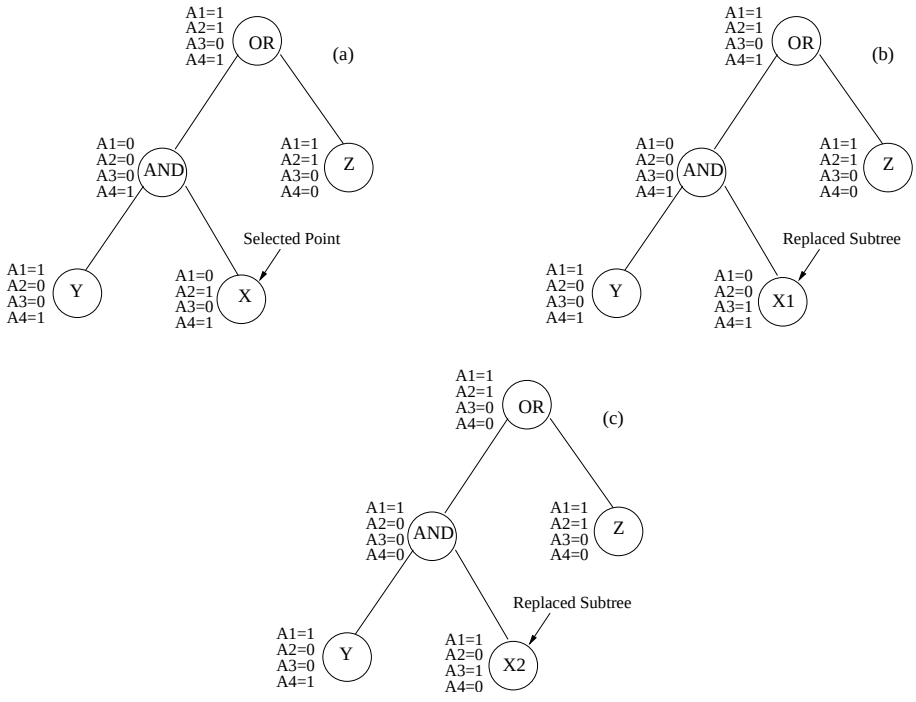

Fig. 11.2: A parent and its generated children from crossover.

at the parent 'AND' node does not change, nor does it change at the root 'OR' node. However when X is replaced with $X2$, the semantics of the ancestors do change.

Thus the form of fixed semantics investigated in McPhee et al. [123] is the only one cause of crossover failing to change semantics. The replacement might simply not change the right components of the semantics for the change to propagate to the root. To remedy this, we propose a crossover operator known as *Guaranteed Change Semantic Crossover* (GCSC). The objective of GCSC is to guarantee a change in the semantics of the children in the new population. The detail of GCSC is presented in algorithm 7.

How can we efficiently compute whether two crossover points are change-inducing? Before we try to exchange X and $X1$ in figure 11.2 (a, b), we compute which attributes differ (in this case, forming the set $\{A2, A3\}$). We then propagate this upward, discovering that at the parent 'AND' node, since Y guarantees that A2 and A3 will be 0, the root difference set R becomes the empty set, ϕ. Therefore, this crossover is not change-inducing, so it is aborted. If we try to exchange X and $X2$ in figure 11.2 (a, c), the difference set is $\{A1, A2, A3, A4\}$, which propagates to the 'AND' as $\{A1, A4\}$, and then to the 'OR' as

Algorithm 7: Guaranteed Change Semantic Crossover

```
select Parent 1 P1;
select Parent 2 P2;
Count=0;
while Count<Max_Trial do
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    if Subtree1 and Subtree2 are change-inducing then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
if Count=Max_Trial then
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    execute crossover;
    return true;
```

$R = \{A4\}$. Since the root change-set is nonempty, this crossover is change-inducing.

The objective of GCSC is similar to that of SDC [11]: to guarantee change. The differences lie in the use of subtree semantics rather than the semantics extracted from using ordered binary decision diagrams (ROBDDs) , and in the checking of only a single child.

11.3.2 Crossover for Improving Semantic Locality

The first crossover used for improving semantic locality is Semantic Similarity based Crossover (SSC). In SSC, the crossover is aborted not only when the two children are semantically too similar, but also when they are too dissimilar. However, for the Boolean domain, since SAC is not guaranteed to change semantics from parents to children, SSC may also work in a similar way and this can diminish the efficiency of SSC.

Here we propose a variant, *Locality Controlled Semantic Crossover* (LCSC). LCSC is an extension of GCSC, bearing the same relation to it as SSC does to SAC. LCSC is

implemented in the same way as GCSC, propagating the difference set of changed attributes upward in the tree. However when the root is reached, the crossover is aborted not only if the difference set is empty, but also if it is too large. In other words, LCSC is only executed if $0 < |\mathrm{R}| \leq \epsilon$, where R is the difference set at the root, and ϵ is a predefined constant called the semantic sensitivity. In this chapter, the value of ϵ is one of the experimental parameters.

11.4 Experimental Settings

This section presents the Boolean problems that we use to test semantic based crossovers and the parameter settings for GP system.

11.4.1 Boolean Problems

We investigated the effects of the new crossover operators GCSC and LCSC, comparing them with SC , SDC [1], SAC and SSC. We tested all operators on four test-bed Boolean problems: 5 bit parity, 5 bit majority, 6 bit multiplexer, and 6 bit parity. These problems have been used in the previous studies on semantic based crossover [123, 13].

The 6-bit multiplexer problem (6MUX) 6MUX interprets the two control bits $\{A_0, A_1\}$ as an address with which to choose the correct input bit from the binary input lines $\{D_0, D_1, D_2, D_3\}$ as its output.

The even 5- and 6-bit parity problems (5PAR and 6PAR) These problems take respectively 5 and 6 bits as input, returning true (1) if and only if an even number of the inputs are true (1).

The 5 majority problem (5MAJ) This problem takes 5 bits as input, and returns true(1) if and only if, the majority of the inputs are true(1).

[1]Here, SDC is implemented using subtree semantics as other crossovers in this chapter rather than using ordered binary decision diagrams as in [11].

In these (minimising) problems, the fitness of an individual is the mean of error bits (the bits that do not match the target function). A run is considered successful if it finds an individual with no error (i.e. fitness zero).

11.4.2 Parameter Settings

The basic parameter settings are similar to the settings in Table 5.2, Chapter 5 with the following modifications.

- The population size is set at 250. This value is calibrated from the experiments to highlight the performance of these crossovers.
- The function set for all problems is {AND, OR, NAND, XOR}
- The terminal set is $\{X_1, X_2, ..., X_N\}$, where N is the number of input variables.

The maximum number of trials permitted to select satisfied subtrees for SDC, SAC, GCSC, SSC and LCSC was set at 20. Although, SDC in [11] repeated attempts at crossover until semantically different children were found. However the effect of continuing beyond 20 attempts is small, while the computational cost can be significant. For better comparability, we used the same bound as for the other algorithms.

The semantic sensitivities for SSC set to 0.2, 0.3, 0.4. These values were tested for real-valued problems and also the values for good performance of SSC for Boolean problems. Three these configurations of SSC will be referred as SSCX with X = 02, 03, 04. Because LCSC testing is more exhaustive than SSC, it is likely to generate change for a given fitness case; thus semantic sensitivity levels need to be set correspondingly lower. We used 0.1, 0.15 and 0.2, for the semantic sensitivities of LCSC. The corresponding treatments being denoted LCSCX with X = 01, 015, 02.

Tab. 11.1: The percentage of successful runs.

Xovers	5PAR	5MAJ	6MUX	6PAR
SC	52	53	15	19
SAC	55	50	19	24
SDC	75	72	16	21
GCSC	67	76	18	22
SSC02	62	60	19	25
SSC03	70	58	22	33
SSC04	71	62	21	41
LCSC01	76	76	48	31
LCSC015	83	70	33	35
LCSC02	84	79	36	47

11.5 Effect of Semantic Diversity and Semantic Locality on GP Performance

To compare the performance of GP under the control of semantic diversity and semantic locality of operators for Boolean problems, we recorded two classic performance metrics, the percentage of successful runs out of 100 runs (table 11.1) and the mean best fitness (table 11.2). We tested the statistical significance of all differences from SC in the results in table 11.2, using a Wilcoxon signed-rank test with a confidence level of 95%. If a crossover operator is significantly better than SC, the result is printed in bold face.

11.5.1 Effects of Promoting Diversity

It can be seen from Table 11.1 that semantic diversity promoting of crossover leads to a positive effect on GP performance on the tested Boolean problems. It is reflected by a greater number of solutions found by semantic based crossovers for enhancing diversity. The table, however, also shows the different effect of these crossovers on different problems. It looks like the positive effect of SAC is not great and on 5MAJ, SAC is even worst than SC in terms of finding solutions. Conversely, the advantage of both GCSC and SDC is greater especially on 5PAR and 5MAJ. These results can be explained when we see the

Tab. 11.2: Mean and Standard Derivation of best fitness. Note that the values are scaled by 10^2.

Xovers	5PAR	5MAJ	6MUX	6PAR
SC	2.24±3.19	1.78±2.18	5.96±3.88	8.32±5.73
SAC	2.56±3.33	2.03±2.40	5.34±3.76	8.12±5.98
SDC	**1.31±2.63**	**0.96±1.64**	5.04±3.51	**6.37±4.93**
GCSC	**1.59±2.71**	**0.84±1.59**	4.68±3.39	7.25±5.47
SSC02	1.96±2.96	1.50±2.05	5.53±4.30	7.53±5.85
SSC03	**1.18±2.07**	1.37±1.68	4.96±3.91	**6.28±5.65**
SSC04	**1.28±2.35**	1.40±1.95	**4.40±3.81**	**5.28±5.32**
LCSC01	**0.96±1.86**	**0.87±1.66**	**3.09±4.26**	**5.95±4.83**
LCSC015	**0.71±1.76**	**1.03±1.58**	**3.39±3.67**	**5.43±5.12**
LCSC02	**0.65±1.61**	**0.75±1.54**	**3.28±3.33**	**4.54±5.45**

percentage of semantics changing from parents to children for each crossover. However, on two more difficult problems, 6MUX and 6PAR, the advantage of both SDC and GCSC seem to be lower.

The results in Table 11.2 are consistent with those in Table 11.1, in that promoting semantic diversity brings a positive influence on the quality of GP found solutions. The table shows that the mean of best fitness of the crossovers for semantic diversity promoting is usually smaller than of SC with only two exceptions on problem 5PAR and 5MAJ for SAC. The margin of the improvement, nevertheless, depends on both problems and the crossover operators. While the enhancement of SAC versus SC seems not so significant, the improvement of GCSC and SDC are consistently greater. This, again can be explained by the ability of each crossover to modify the semantics of its parents.

The statistical test results show the less convincing better of SAC over SC. SAC is not significantly better than SC on any problem. By the contrast, both GCSC and SDC are significantly better than SC on two easier problems (5PAR, 5MAJ) and only SDC is significantly better than SC on 6PAR. In general, the results in this section provide evidence for the positive effect of the crossovers for semantically promoting diversity. However, on problems 6MUX, none of these crossovers (namely SAC, SDC and GCSC) help to significantly improve the performance of GP in comparison with SC.

11.5.2 Effects of Improving Locality

Table 11.1 shows how effective locality contributes to GP performance. It can be recognised that keeping a semantically small change leads to a further improvement of a GP system in comparison with only promoting semantic diversity. Generally, SSC is better than SAC and LCSC is superior to GCSC. This table shows that SSC not only works well with real valued problems, as has been shown in the early chapters, but also works well with Boolean problems. It, however, also shows that in Boolean problems, a better way for designing an operator may exist and LCSC is a typical instance. The table clearly shows LCSC has the best performance in comparison with other tested crossovers.

Table 11.2 is consistent with Table 11.1, it confirms the further improvement of performance when improving locality. It can be observed that controlling locality as in SSC leads to a more frequently significant improvement versus SC in comparison with purely promoting diversity as in SAC. Likewise, LCSC is usually better than GCSC and always significantly better than SC. On problems 6MUX, where none of the crossovers for promoting semantic diversity make significant improvement, both crossovers for enhancing semantic locality significantly improve GP performance and LCSC makes significant improvement on all the tested semantic sensitivities.

All in all, the results in this section again demonstrate that semantically promoting diversity is only one part of a story and that improving semantic locality is even more important. It is strongly believed that combining enhancing semantic diversity and improving locality helps to further improve GP performance and intensively promoting diversity and suitably keeping semantically small change as in LCSC may be better than other purely semantic diversity controlled methods.

11.6 Some Properties of Semantic based Crossovers

This section presents some behavioural aspects of GP under semantic based crossovers. The properties analysed in this section are: the rate of crossover events that exchange two

Tab. 11.3: Average percentage of semantically equivalent subtrees in crossover.

Xovers	5PAR	5MAJ	6MUX	6PAR
SC	6.89	7.63	6.22	5.72
SAC	0.01	0.01	0.01	0.01
SDC	0.22	0.12	0.24	0.31
GCSC	0.35	0.75	0.20	0.07
SSC02	5.80	5.53	5.22	4.96
SSC03	1.35	0.99	1.34	1.38
SSC04	0.43	0.40	0.48	0.49
LCSC01	2.55	2.30	1.61	1.24
LCSC015	1.71	1.23	0.82	0.69
LCSC02	1.49	1.01	0.60	0.51

semantically equivalent subtrees, the rate of crossover events that change the semantics and fitness from the parents to the children, the locality (the movement step of both semantics and fitness of the individuals) property of these crossovers, the crossover constructive effect, and the effect of these crossovers on code bloat of the GP systems.

11.6.1 Semantic Equivalence

The first result recorded the extent of semantically equivalent exchanges arising from the tested crossover operators. Since the proposed new crossover operators (SAC and SSC, GCSC, SDC, and LCSC) work by analysing the semantics of subtrees, and tries to prevent the exchange of semantically equivalent subtrees, it would be informative to see how frequently this actually happens. This information shows us how frequently SC fails to change the semantics of individuals (semantically unproductive) and the extent to which SAC, SSC, GCSC, SDC and LCSC could solve this problem. The statistics collected are the percentage of such crossover events, for all crossovers. The results are shown in Table 11.3.

The table shows that there is quite small portion of crossover that exchanges two semantically equivalent subtrees. This value for SC is only from 6% to 7%. This proves that exchanging semantically equivalent subtrees is not the main reason that causes semantics to

Tab. 11.4: The percentage of semantic change and fitness change of crossovers.

Crossovers	5PAR		5MAJ		6MUX		6PAR	
	SmD	FnD	SmD	FnD	SmD	FnD	SmD	FnD
SC	25.7	21.6	24.6	18.5	29.0	22.9	33.6	27.5
SAC	27.5	22.1	25.8	19.0	29.8	23.4	34.5	28.9
SDC	94.4	74.0	90.2	64.6	96.3	74.6	98.9	74.4
GCSC	95.4	74.5	91.3	64.7	97.2	75.4	98.8	74.4
SSC02	25.7	20.7	23.6	18.2	27.2	22.8	34.6	28.1
SSC03	23.4	19.2	21.3	16.5	26.8	21.4	32.5	26.7
SSC04	24.3	20.5	22.4	17.6	28.4	21.9	33.2	17.4
LCSC01	73.8	57.6	81.3	58.1	85.8	64.5	86.9	60.4
LCSC015	80.9	61.4	89.4	62.7	93.2	70.5	92.1	65.4
LCSC02	83.4	63.4	91.2	64.4	94.9	72.8	93.8	67.6

be unchanged from parents to children, but the replacement of inapproriate subtrees could be the main reason. The table also shows that semantic based crossovers successfully prevent the exchange of two semantically equivalent subtrees. The percentage of this phenomenon with semantic based crossovers is often less than 3% except on SSC02. It should be noted that the values for crossovers which improve semantic locality is often greater than crossovers for promoting semantic diversity. The reason is that SSC and LCSC also have to focus on finding subtrees that make a semantically small change.

11.6.2 Semantic Diversity

The second statistics recorded the rate of crossover events that change the semantics from parents to children. Notice that, the change of semantics does not lead to the change of fitness from parents to children (for Boolean problems, there may be many individuals that have different semantics but the same fitness). Therefore a statistic that recorded the percentage of crossovers that change the fitness from parents to children is also recorded.

The rate of crossover events that changes semantics (Semantic Difference - SmD) and fitness (Fitness Difference - FnD) is shown in Table 11.4. It can be seen from this table that the proportion of SC that can change semantics from parents to children is rather

small. There is only about from 20% to 30% of SC that change semantics from parents to children. It is different from the real valued problems where around 60% to 80% of SC that can change semantics. It explains why promoting semantic diversity in real valued problems has less effect than in Boolean problems. The small percentage of SC that can modify the semantics leads to a small percentage that can alter fitness. This value is about 18% to 25%.

By preventing the swap of two semantically equivalent subtrees, SAC slightly improves the ability of crossover in generating semantically different children. These results explain why the advance of SAC over SC is less convincing. These values of SSC are also nearly the same as those of SC and in fact SSC often produces fewer semantically different children from their parents. It is, therefore, assumed that the noticeable enhancement of SSC versus SC is mostly due to its locality property.

Conversely, three crossovers, SDC, GCSC and LCSC are much better than the other crossovers in terms of changing semantics and fitness after crossover. This means that these crossover have achieved their objective in terms of promoting semantic diversity. Hence, this is also evidence for the remarkable improvement of these three crossovers over SC.

11.6.3 Semantic Locality

The next set of statistical results are used to analyze the locality property of the crossovers. Since, the main aim of SSC and LCSC is to improve the locality of these crossovers in comparison with SC, it is very important to see how successful they are at enhancing locality. To compare the locality of these crossovers an experiment was conducted where the semantic change and fitness change of individuals before and after crossover was measured. For example, suppose two individuals having semantics of 00101010 (P1) and 10110101 (P2) are selected for crossover and after crossover their semantics are 10100100 (C1) and 00101001 (C2) respectively. Then the change of semantics from the parents to the children is 5 with P1, C1 and 4 with P2 and C2. In total, the change of semantics from the parents to the children is $(5 + 4)/(8 * 2) = 9/16 = 0.56$.

Tab. 11.5: The size of semantic change and fitness change of crossovers. Note that the values are scaled by 10^2.

Crossovers	5PAR		5MAJ		6MUX		6PAR	
	SmC	FnC	SmC	FnC	SmC	FnC	SmC	FnC
SC	15.7	12.5	11.7	8.75	11.4	7.78	14.1	9.28
SAC	15.9	12.8	11.7	8.71	11.3	7.66	14.3	9.54
SDC	14.4	11.8	11.6	8.08	11.4	7.46	13.2	8.49
GCSC	14.3	11.4	11.6	7.90	11.1	7.26	13.3	8.58
SSC02	14.4	11.2	9.18	6.72	9.52	6.22	12.4	8.18
SSC03	12.0	10.2	7.96	6.09	8.41	6.01	10.4	7.60
SSC04	12.4	10.3	8.46	6.68	8.58	6.31	10.7	7.95
LCSC01	9.15	7.93	7.19	5.29	6.50	4.61	6.90	5.11
LCSC015	9.32	8.09	7.25	5.32	6.86	4.82	7.20	5.11
LCSC02	10.2	8.20	8.16	5.34	7.47	5.17	7.98	5.53

Similarly, if the parents have fitness of 0.75 and 0.45 are selected for crossover, and after the crossover operation their children have fitness of 0.80 and 0.15 The change of fitness of these individuals is $(Abs(0.75-0.8)+Abs(0.45-0.15))/2 = 0.175$. These values (Semantic Change - SmC, and Fitness Change - FnC) were averaged over the whole population and over 100 runs. The average semantic change and fitness change of individuals before and after crossover is shown in Table 11.5 (the values are scaled by 10^2).

The table shows that there is very small difference between SAC, SDC, GCSC versus SC in terms of the size of both semantic and fitness change. It is reasonable as these crossovers do not aim to control locality. By contrast, the table confirms that the size of the semantic change and the fitness change of both SSC and LCSC were remarkably smaller than SC, SAC, SDC or GCSC. These results lead to a smoother change both in semantics and fitness over the generations of SSC and LCSC versus SC, SAC, SDC and GCSC. The table also shows that the crossovers for improving semantic locality have achieved their objective in making a smaller change in terms of semantics from parents to children.

Tab. 11.6: Semi-constructive and full-constructive effect of crossovers.

Crossovers	5PAR		5MAJ		6MUX		6PAR	
	SeC	FuC	SeC	FuC	SeC	FuC	SeC	FuC
SC	2.51	0.15	2.12	0.14	3.61	0.24	4.33	0.29
SAC	2.61	0.18	2.22	0.16	3.86	0.28	4.59	0.32
SDC	20.2	1.96	15.7	1.15	21.2	1.87	21.6	1.92
GCSC	20.3	1.71	16.0	1.11	21.4	1.90	21.4	1.90
SSC02	2.67	0.19	2.31	0.18	3.94	0.33	4.61	0.36
SSC03	2.15	0.20	2.34	0.19	3.90	0.34	4.46	0.34
SSC04	2.16	0.18	2.35	0.17	3.95	0.35	4.81	0.39
LCSC01	14.8	1.32	13.6	1.02	18.3	1.66	17.0	1.59
LCSC015	16.0	1.37	15.3	1.09	21.2	1.82	19.6	1.76
LCSC02	16.8	1.82	16.8	1.24	21.6	1.92	20.7	1.98

11.6.4 Constructive Effect

The next set of statistical results examine the constructive effect of the crossovers on Boolean problems. The early results show that SDC, GCSC and LCSC are more semantically productive than the others crossovers and that SSC and LCSC have higher locality than all other operators. This leads to a further question, whether these properties help the crossover operators to frequently generate better children than their parents (more constructive crossover). In other words, we would like to know the relative constructiveness of the crossovers. Statistics were collected measuring the constructive effect of these crossovers using a method similar to that which has been used in Chapter 7. Here we also distinguish two types of constructive crossover: semi-constructive and full-constructive crossover.

The percentage of semi-constructive (SeC) events and full-constructive (FuC) events of these crossovers are shown in Table 11.6. It can be seen from this table that SAC and SSC is only sightly more semi-constructive and full-constructive than SC, while SDC, GCSC and LCSC are more constructive than the other operators. The reason may be that SAC and SSC are only slightly more semantically productive than SC, whereas SDC, GCSC and LCSC are remarkably more semantically productive than SC. These results support

Tab. 11.7: The average of individual size in the population and the average size of solutions.

Crossovers	5PAR		5MAJ		6MUX		6PAR	
	Pop	Sol	Pop	Sol	Pop	Sol	Pop	Sol
SC	177.2	207.8	133.4	218.3	144.4	212.1	176.7	195.5
SAC	186.2	215.9	137.6	228.8	148.6	203.6	178.8	206.7
SDC	195.6	219.7	138.9	274.7	161.8	320.0	193.9	223.6
GCSC	197.0	237.3	135.7	267.2	167.7	247.1	199.1	245.8
SSC02	168.4	193.4	130.2	201.3	142.2	220.7	172.4	144.7
SSC03	157.4	167.3	123.8	185.0	129.0	151.7	156.8	157.7
SSC04	166.9	173.3	125.8	215.3	132.1	131.4	163.3	143.7
LCSC01	145.6	175.7	109.9	213.3	115.3	196.1	158.1	140.1
LCSC015	165.4	146.0	110.3	197.6	117.5	181.5	144.3	170.0
LCSC02	150.3	158.5	107.3	217.5	128.5	254.8	141.2	162.9

the improvement of these crossovers in comparison with other operators.

11.6.5 Code Bloat Effect

The last set of statistics investigated the effect of the crossovers on GP code boat. It has been shown in the Chapter 7 that promoting semantic diversity leads to slightly more bloat while improving semantic locality leads to reduce code bloat for real valued problems. Therefore, it would be very interesting to see how semantic based crossovers affect to code bloat on Boolean problems?

To investiage the effect of these crossovers on code bloat we measure the average size of individual (number of nodes) in the population over 50 generations, averaged over 100 runs. This statistics is presented in Tables 11.7 (Pop column). The table also shows the average size of the solutions (the best fitness individuals) found by each crossovers (Sol column).

The results in this table are consistent with the results in the Chapter 7. Again, it can be seen that promoting semantic diversity of crossover leads to a little more bloat than SC. The table shows that the average size of the population of SAC, SDC and GCSC are greater than SC. In these three crossovers, it seems that SAC has less bloat than SDC and

GCSC. The reason could be that SAC does not promote semantic diversity as extensively as SDC and GCSC.

While promoting semantic diversity exacerbates code bloat, improving semantic locality always helps to reduce code bloat. The average individual size in the population of both SSC and LCSC are consistently smaller than one of SC. The table also shows that the size of reduction code bloat of LCSC is often greater than SSC. The reason could be that LCSC is often more successful than SSC in changing semantics from parents to children.

The diminishing code bloat of SSC and LCSC potentially help these crossovers to find more comprehensive solutions (smaller and more readable solutions). This is confirmed by the results in Table 11.7 (Sol column). It can be observed from this table that both SSC and LCSC usually find solutions with better quality in terms of size. The exception happens only in one case on 6MUX problems with LCSC. Conversely, The exacerbating bloat of SAC, SDC and GCSC often lead them to find bigger solutions in terms of size. Generally, this section shows the positive effect of improving semantic locality and slightly negative of promoting semantic diversity of crossover on GP code bloat.

11.7 Conclusions

This chapter investigated the impact of semantic based crossovers on Boolean problems. For Boolean problems, some novel semantic based crossovers were introduced. Again, these crossovers addressed two main objectives: promoting semantic diversity and improving semantic locality. The semantic based crossovers were tested on four test bed Boolean problems. The results showed that promoting semantic diversity helps to improve GP performance, but improving semantic locality is even more beneficial.

Next, some properties of the GP system under the effect of these crossovers were analysed. The results showed that the designed crossovers achieved their objectives in promoting semantic diversity and improving semantic locality. This led to a positive effect on constructive rate, a positive impact of SSC and LCSC on GP code bloat and negative impact of SAC, SDC and GCSC on the code bloat effect of GP. Overall, this chapter showed

that improving semantic locality combining with promoting semantic locality of crossover is not only important for real valued problems but also vital for Boolean problems. In the following chapter we go on to conclude the thesis.

Chapter 12

Predicting Time Series using Semantic based Crossovers

In this chapter, semantic based crossovers are applied to time series problems. We compare the prediction ability of two semantic based crossovers – Semantic Similarity based Crossover (SSC) and the Most Semantic Similarity based Crossover (MSSC) with standard crossover on two time series. The first time series is an artificial time series (Mackey series) and the second one is a real time series (Tide series in Venice Lagoon, Italy). The results show the comparative ability to predict of semantic based crossovers versus standard crossover. Some results in this chapter have been published in [138].

12.1 Introduction

The previous results have shown the advantages of using semantic based crossovers in comparison with standard crossovers on not only real valued problems but also on Boolean problems, on not only GP performance but also on the ability of GP to generalise. However, the tested problems in the preceding chapters are artificial problems. This raises a question whether the advantages of semantic based crossovers is maintained when solving a real world problem.

In the realm of GP real world applications, time series prediction has been considered as a major target [77, 172, 165, 110]. The main approach in the previous work is to build up forecasting models by combining different variables that can represent the knowledge from time series data.

In this chapter, we use two semantic based crossovers (SSC and MSSC) to solve the problem of predicting time series. The first time series is an artificial time series (Mackey series) and the second one is the tide series in Venice Lagoon, Italy. Two these problems has been seen as truly difficult problems [1, 73].

12.2 Predicting Mackey Time Series

This section presents the ability of semantic based crossovers to predict Mackey time series. First, we clearly state the problem. The experimental settings are given next. After that, results are presented and discussed.

12.2.1 Problem Statement

Mackey and Glass introduced their time series in 1977 [114]. It has been widely used as a benchmark for the generalisation ability of a variety of machine learning methods. The Mackey-Glass equation is:

$$x(t+1) = x(t) - bx(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{12.1}$$

With $x(0) = 1$, $a = 0.2$, $b = 0.1$ and $\tau = 17$, the time series is aperiodic, non-convergent, and completely chaotic. Figure 12.1 plots the 500 points (from position 3501 to 4000) of Mackey time series.

The requirement in time series prediction is to estimate the future values of a series

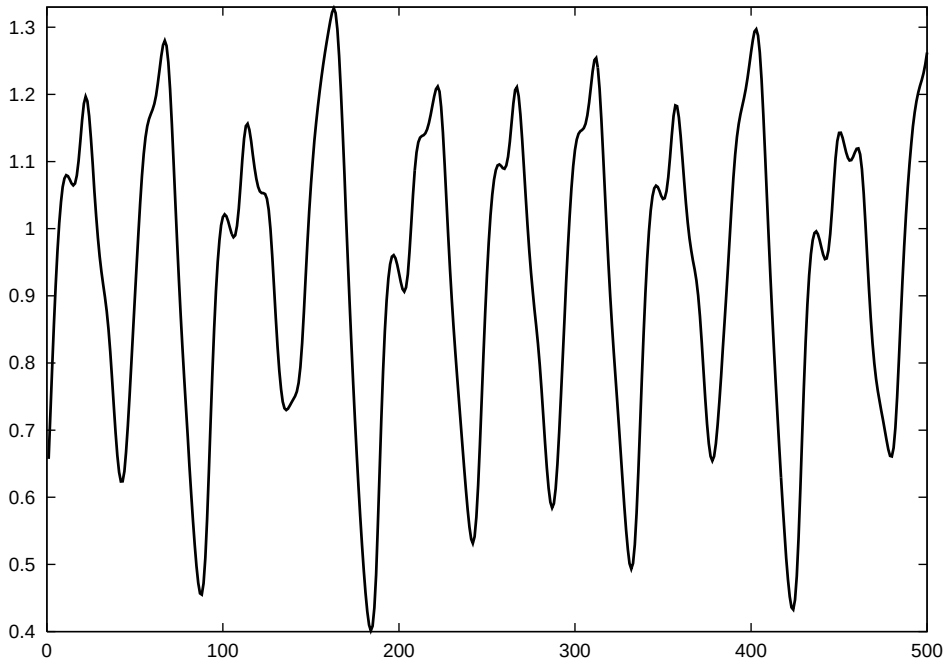

Fig. 12.1: Mackey-Glass 500 points (from position 3501 to 4000) ($a = 0.2$, $b = 0.1$ and $\tau = 17$).

based on its past values. In other words, it is to find a function F such that:

$$x(t+1) = F(x(t), x(t-\alpha), ..., x(t-N\alpha)) \tag{12.2}$$

This problem is generally considered quite tough, as there is no exact closed form solution [73]. We follow the approach of [59] to discard 3500 initial points of the series in order to avoid initialisation transients and set $\alpha = 1$ and $N = 8$ as in [159]. That is, our task is to estimate the value of the series at time $t + 1$ given the 8 values at times $t, t-1, ..., t-7$ in the past.

12.2.2 Experimental Settings

Our experiment compared the ability of GP systems with standard crossover and two semantic based crossovers to solve the Mackey-Glass problem. The experimental settings are as in Table 5.2, the only difference being the terminal set, which here consists of

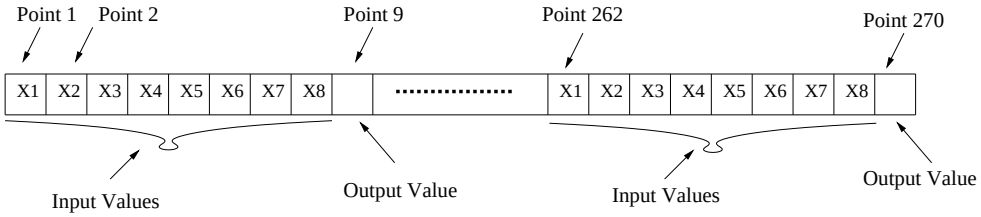

Fig. 12.2: The plot of 270 points that are combined into 30 fitness cases.

one constant (1) and 8 variables $(X_1, X_2, ..., X_8)$ representing the time series values at $x(t), x(t-1), ..., x(t-7)$. The lower semantic sensitivity of SSC is set to 10^{-3} and upper bound set to 0.4. Three value of Max_Trial, 12, 16, and 20 are tested (They are the values that have been used in the previous experiments). These configurations of SSC will be referred as SSCX with X=12, 16, 20. Similarly, three configurations of MSSC are tested and they are shorthanded in a similar way to SSC.

We took the first 370 points of Mackey-Glass time series as the data set. The first 270 points were used for constructing the training set. These points are divided into 30 blocks of 9 consecutive points (indexed from 1 to 9). For each block, a fitness case for the training set is extracted as follows. The point at position 9 is the output and the 8 points at positions 8, 7, ..., 1 are the inputs. Figure 12.2 is the plot of 270 points that are combined into 30 fitness cases

Three test sets were used, referred to as Test k with $k = 0, 1, 2$. Each test set is extracted in the same way as above from a set of 270 points, but for Test k, these consist of the last $270 - 10^k$ points of the first 270 points, together with the first 10^k points of the remaining 100 points of the series. For example, with $k = 2$, we use the points from 101...270 (the last 170 points of the first 270), together with the following 100 points, giving a total of 270 points. For each such set of 270 points, we extract the test set in the same way as for the training set.

To investigate the impact of population size to the ability of GP in solving this problem, in this experiment, 3 population sizes of 250, 500 and 1000, were tested. Correspondingly, three values of generation: 100, 50, 25 respectively, were used. These values fixed the number of fitness evaluations as 25000.

Tab. 12.1: The mean best fitness on the training sets and on the three testing sets. Note that the values are scaled by 10^2.

Crossovers	Pop250				Pop500				Pop1000			
	Tr	T0	T1	T2	Tr	T0	T1	T2	Tr	T0	T1	T2
SC	1.03	1.05	1.11	1.06	0.82	0.86	0.92	0.82	0.76	0.78	0.80	0.71
SSC12	**0.81**	**0.93**	**0.98**	**0.88**	**0.69**	0.80	0.86	0.77	**0.67**	**0.70**	0.78	0.68
SSC16	**0.86**	1.01	1.01	**0.90**	**0.64**	0.82	**0.82**	0.77	**0.64**	0.72	**0.73**	0.67
SSC20	**0.80**	**0.88**	**0.97**	**0.88**	**0.68**	**0.76**	**0.83**	**0.75**	**0.64**	**0.70**	**0.72**	**0.65**
MSSC12	**0.79**	**0.84**	**0.88**	**0.80**	**0.58**	**0.64**	**0.68**	**0.62**	**0.57**	**0.63**	**0.66**	**0.61**
MSSC16	**0.78**	**0.84**	**0.89**	**0.80**	**0.64**	**0.71**	**0.74**	**0.72**	**0.57**	**0.72**	**0.72**	**0.62**
MSSC20	**0.77**	**0.87**	**0.90**	**0.82**	**0.61**	**0.69**	**0.72**	**0.66**	**0.57**	**0.63**	**0.66**	**0.60**

12.2.3 Results

In order to compare the performance and the ability to predict of these crossovers, we recorded the best fitness on the training set. This value is then averaged for 100 runs. Also, the individual with the best fitness on the training set is selected and tested on the three testing sets. These values are then again averaged over 100 runs. The best fitness on the training set (Tr column) and the best fitness on three testing sets (T0, T1, T2 columns) are shown in Table 12.1. We also tested the statistical significance of all differences from SC in the results in table 12.1, using the Wilcoxon signed-rank test with a confidence level of 95%. If a crossover operator is significantly better than SC, the result is printed in bold face.

12.2.4 Discussion

We first look at the performance of semantic based crossovers compared to standard crossovers. Obviously, these semantic based crossovers (SSC and MSSC) help GP to improve its performance in solving Mackey time series problems. It can be seem from this table that the best fitnesses found by SSC and MSSC are always smaller than ones found by SC. The statistical test shows that the improvement of SSC and MSSC over SC is always significant.

We now compare the prediction ability of these crossovers. It can be seem from Ta-

ble 12.1 that semantic based crossovers not only help to improve GP performance on the training set but also on the testing sets. The table shows that the mean error on the testing sets of SSC and MSSC is always smaller than one of SC. The statistical test show that the improvement on the testing sets of SSC and MSSC over SC is often significant with only some exceptions on SSC. In comparison between SSC and MSSC the table again shows that MSSC is superior to SSC both on the training test and the testing sets.

Comparing between three testing sets, the results show that, in this problem, predicting long time does not necessary more difficult than predicting short time. Definitely, the error on the two first testing sets (T0 and T1) is often greater than one on the third testing set (T2) while T1 is also greater than T0. This result is not entirely surprising since Rivero et al. also shows that the error of predicting long time can be smaller than predicting short time [165].

In terms of the effect of population size on the GP performance and GP ability to predict of these crossovers, it can be seen that a bigger population with a shorter generation brings a better result than a smaller population with a longer generations on both GP performance and the ability of GP to predict. It is reflected by the smaller values of mean best fitness on the population of 1000 versus the population of 500 and 250 and of the population of 500 versus 250. It should be noted here that all systems have the same number of fitness evaluation (25000), therefore using a bigger population size is more beneficial in solving this problem.

12.3 Predicting Tide

This section presents the ability of using semantic based crossovers in predicting the Tide in Venice Lagoon, Italy. The description of the problem is given first. Next are experimental settings, results and some discussion.

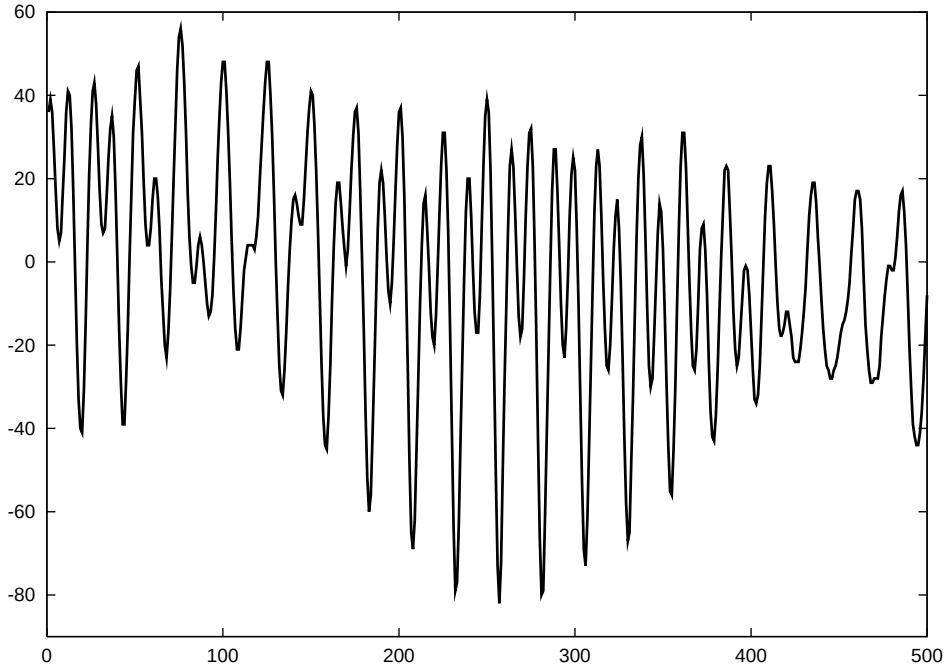

Fig. 12.3: The plot of 500 first points of the tide level in Venice Lagoon.

12.3.1 Problem Statement

The prediction of high tides has long been the subject of interest to humans. The motivation for such interest is the economic benefits of the prediction/forecasting system. Usually, high tides is the result of a combination of some chaotic climatic elements. Therefore, tide's behaviour is difficult to predict, because it depends on too many factors [1]. Two main factors that most affect tide level are the astronomic and atmospheric agents. The problem has been approached by using time series analysis and nonlinear neural networks [81]. In this experiment we use GP equipped with semantic-based crossovers to solve the problem.

The time series data to predict in this section is high tides in Venice Lagoon, Italy. The data is measured in centimeters each hour along the years 1990 to 1995 [1]. Figure 12.3 plots 500 first points of this data [1]. It has been shown in [81] the tide is period of 24 hours. Therefore, the authors predict the value in the future based on the values of 24 successively previous points. However, they argued that this model have a large number of input values

[1]To make it easier for GP to solve this problem, we normalise data to the range from -1 to 1.

and it makes the problem more difficult to be predict. Therefore, they simplify the model by reducing the number of input values from 24 to 7. In other words, their model is to find function F so that.

$$x(t+1) = F(x(t), x(t-4), ..., x(t-24)) \tag{12.3}$$

However, we have conducted an experiment to predict the tide with this model and we found that it is more difficult to predict (the error on the testing sets is greater) than the following model:

$$x(t+1) = F(x(t), x(t-1), ..., x(t-6)) \tag{12.4}$$

Therefore, in this experiment we predict the tide in the future based on 7 successive values in the past as the above equation.

12.3.2 Experimental Settings

The experimental settings for GP are as the same in Table 5.2, the only difference being the terminal set, which here consists of one constant (1) and 7 variables $(X_1, X_2, ..., X_7)$ representing the time series values at $x(t), x(t-1), ..., x(t-6)$. Three configurations of SSC and MSSC are similar to those in the previous section are used and have the same naming convention.

We took 340 points (from position of 1 to position of 340) from this data set. We divided these data into two sets, one for training and the other for testing. The first 240 values are used for the training set. They are combined into 30 fitness cases. Similarly, 30 fitness cases were used for the testing set. We divided the testing set into 3 sets with the number of points in the future is 10^k $(k = 0, 1, 2)$. These 3 sets will be referred as Tk with $k = 0, 1, 2$ in the following text.

Tab. 12.2: The mean best fitness on the training sets and on the three testing sets. Note that the values are scaled by 10^2.

Crossovers	Pop250				Pop500				Pop1000			
	Tr	T1	T2	T3	Tr	T1	T2	T3	Tr	T1	T2	T3
SC	5.58	12.5	7.33	7.39	5.30	10.9	8.84	7.60	5.35	9.98	6.42	6.72
SSC12	5.37	**10.7**	7.21	**6.44**	5.20	8.99	**6.47**	**5.99**	5.03	**7.45**	6.16	**5.35**
SSC16	5.47	11.9	7.16	6.74	4.87	9.32	**6.48**	**6.73**	**4.81**	**7.36**	**5.95**	**5.40**
SSC20	5.45	11.8	7.26	7.36	**4.84**	**8.84**	**6.31**	**6.52**	**4.74**	**7.18**	**5.75**	**5.09**
MSSC12	5.42	11.0	7.03	7.06	**4.39**	**8.30**	**5.99**	**5.77**	**4.47**	**6.77**	**5.30**	**5.34**
MSSC16	5.52	**10.8**	7.19	**6.60**	**4.69**	**9.03**	**6.59**	**6.08**	**4.25**	**6.46**	**4.91**	**4.76**
MSSC20	5.35	**10.5**	**6.61**	**6.39**	**4.65**	9.22	**6.04**	**6.30**	**4.21**	**6.82**	**5.11**	**5.01**

12.3.3 Results

Similar to the previous section, we recorded the best fitness on the training set. This value is then averaged for 100 runs. Also, the individual with the best fitness on the training set is selected and tested on three testing sets. These values are then again averaged over 100 runs. The best fitness on the training set (Tr column) and the best fitness on three testing sets (T0, T1, T2 columns) are shown in Table 12.2. We also tested the statistical significance of all differences from SC in the results in table 12.2, using a Wilcoxon signed-rank test with a confidence level of 95%. If a crossover operator is significantly better than SC, the result is printed in bold face.

12.3.4 Discussion

On the GP performance, we can see from the table that semantic based crossovers is better than standard crossover in solving this problem. The mean best fitness of SSC and MSSC is always smaller than one of SC. The statistical test result shows that the improvement of SSC and MSSC over SC is often significant with exception on the population of 250 with both SSC and MSSC and some cases of the population of 500 and 1000 with SSC.

On the ability to predict of these crossovers, the table shows semantic crossovers again help GP to improve its prediction ability. The error on the testing sets is always smaller with SSC and MSSC compared to SC. The Wilcoxon test results shows that SSC and

MSSC often help to significanly improve the ability of GP to predict versus SC in solving this problem. The table also shows that MSSC is better than SSC. The values found by MSSC is often smaller and more often siginicant than SSC versus SC.

Comparing between three testing sets, the results show that predicting long time is easier than predicting short time. The error found on T2 is often smaller than on T1 and on T1 is often smaller than on T0. On the impact of there popolation size. The table again shows that a bigger population and a shorter generation is better than a smaller population and a longer generation. What is important here is on the biggest population (the best configuration to predict in this problem), MSSC is always significant better than SC and SSC is more usually significant better than SC.

12.4 Conclusion

This chapter applied semantic based crossovers to two time seriec prediction problems. The first time series is an artificial Mackey time series and the second one is the tide series in Venice Lagoon, Italy. The results on both problems showed that semantic based crossovers, especially the Most Semantic Similarity based Crossover (MSSC), often significantly improve both GP performance and GP ability to predict. These results showed the ability of semantic based crossovers in solving real world problems.

Chapter 13

Conclusions and Future Work

This chapter first gives a summary of the main contributions of the thesis and then some possible future extensions originating from the thesis are outlined.

13.1 Contributions

In addition to a review of literature regarding the research in the thesis, the following contributions can be drawn from the investigations presented in this thesis.

A novel way to measure semantics: The thesis proposed a new way to measure semantics in Genetic Programming (GP). This semantics was called *Sampling Semantics*. The sampling semantics of a subtree was estimated by evaluating the subtree on a number of points that are sampled from the problem domain. This way of measuring semantics provides us an easy way to approximate the semantics of a subtree in real valued problems, which is often difficult to measure by using other methods.

New semantics relationships: Based on sampling semantics, a semantic distance was defined as the mean of the absolute distance of semantic points. From that two semantics relationships between two subtrees were introduced. The first semantic relationship is Semantic Equivalence and the second relationship is Semantic Similarity. These relationships support the design of semantic based operators.

New semantic based operators for promoting semantic diversity: By analysing the semantic relationship between subtrees in operators, the thesis proposed a crossover, Semantic Aware Crossover (SAC) and a mutation, Semantic Aware Mutation (SAM) to promote semantic diversity. These operators were executed only if the exchanged subtrees were not semantically equivalent. The experimental results showed the positive impact of these operators on GP performance.

New semantic based operators for improving semantic locality: Based on the semantic similarity relationship, the thesis proposed two semantic based operators for improving semantic locality. The first operator was Semantic Similarity based Crossover (SSC) and the second one was Semantic Similarity base Mutation (SSM). These operators improved semantic locality of standard operators by making a smaller change, in terms of semantics, from parents to children. The results from the experiments showed that they led to a further improvement of GP performance in comparison with the operators for promoting semantic diversity.

A new GP tree-based representation: Since it takes time to evaluate the sampling semantics of a subtree, the thesis introduced a new GP tree-based representation to overcome this drawback. First, the values for qualifying semantics were moved from random points to the fitness cases of the problem. Then, the new tree-based representation was formed by adding a number of attributes to every node in the traditional tree-based representation. These attributes were used to store the semantics of each subtrees in a GP individual. This representation helped to speed up the execution of semantic based crossovers.

Two methods to improve Semantic Similarity based Crossover: Although, the performance of SSC was better than some other tested crossovers, it had a weakness: its performance depended on some tunable parameters (semantic sensitivities). To remedy this, the thesis proposed two approaches. The first method self-adapted SSC's semantic sensitivities and the second one implicitly removed its higher bound of semantic sensitivities. These methods not only overcame SSC's drawbacks but also helped to improve its performance.

Experimental analysis of properties of semantic based crossovers: To understand the reason behind the improvement of semantic based crossovers, a thorough analysis on some crucial properties of GP under semantic based crossovers was conducted. The properties that were analysed include semantic diversity, semantic locality, the constructive effect of crossovers and their code bloat effect. These analyses helped to explain the superior performance of semantic based crossover versus other crossovers.

An Investigation on the generalisation ability of semantic based crossovers: Since, generalisation has long been seen as one of the most desirable properties of a machine learning method, the thesis investigated the ability of semantic based crossovers to generalise. The investigation showed that the superior performance of semantic based crossover was maintained on unseen data.

A comparison between the role of semantic and syntactic locality of crossover: Locality is important for all search methods. However, is semantic locality or syntactic locality more important? To address this question, the thesis proposed a new syntactic based crossover for improving syntactic locality and compared it with the semantic based crossover for enhancing semantic locality. The results showed that semantic locality is more important in improving GP performance.

A study of semantic based crossovers with problem difficulty: Although semantic based crossovers helped to improve the performance of GP, would they still maintain their superior performance when problems become harder? The thesis addressed that question by comparing the performance of semantic based crossovers with standard crossover on a class of increasingly difficult problems. The results showed that semantic based crossovers could deal well with increasingly difficult problems.

A study of semantic based crossovers and fitness landscape: Is there any relationship between the fitness landscape of a problem and the semantic locality of an operator? The thesis investigated that relationship by conducting an analysis of the fitness landscape of semantic based crossovers for improving semantic locality. The results in Chapter 10 showed that there is a strong relationship between the fitness landscape of a problem with the semantic locality of operators and that improving semantic locality led

to a smoother fitness landscape.

An extension of semantic based crossover to Boolean problems: Finally, this thesis applied semantic based crossovers to Boolean problems. For Boolean problems, some novel semantic based crossover were proposed. These operators again addressed the problem of promoting semantic diversity and improving semantic locality. The results in Chapter 11 showed that both semantic diversity and semantic locality were important for Boolean problems and that the latter led to further improvement of GP performance. In addition, a demonstration of the success of semantic based operators can also be seen on other problems domain, namely time series prediction problems, in Chapter 12.

13.2 Future Work

Building upon this research, there are numerous avenues for future work. Several of these avenues are discussed here.

First, the various semantic based mechanisms interacted quite simply with standard crossover, just rejecting some proposed exchanges. Thus other crossovers – crossover with bias on the depth of nodes [79] or one point crossover and uniform crossover [156, 108] – could be used instead. In the near future we plan to combine them with semantic based crossovers, especially with SSC, SASE and MSSC.

Next, the semantic based crossovers can also be used with Linear Scaling [92]. Linear scaling is a simple technique that is used for smoothing out the fitness function, which is measure by mean squared error (MSE) on training data [78]. Future work could combine linear scaling with semantic based crossovers to see if they help to further improve GP performance in solving symbolic regression problems.

Another potential research direction is to apply semantic based crossovers to other problem domains. Although, this thesis has applied semantic based crossovers to real valued symbolic regression problems, Boolean problems and time series prediction, and the results show the ability of these crossovers in solving these kind of problems, the question if these crossovers can be applied to other domain problems such as classification

problems, control strategy problems, ect. is still open. For such problems we suspect that semantics must be measured differently and semantic based crossovers have also to be differently designed.

This thesis showed that semantic based crossovers, SSC, SASE and MSSC work to reduce code bloat, but the reason behind this reduction is still unclear from the thesis. Future work aims to answer the question why these crossovers reduce GP code bloat. One direction is to apply a similar method of Soule [178] to study the effect of semantic based crossovers on code bloat during the evolutionary process and the role of various branch size in these crossovers.

Similarly, the thesis showed that semantic based crossover help to improve the generalisation ability of GP, however the reason why the ability to generalise of semantic based crossovers better than standard crossover is not investigated in this thesis. One may suspect the reduction in GP code bloat of these crossovers might help them improve generalisation ability. Nevertheless, the thesis has indicated that controling code bloat does not necessarily lead to enhancing the ability to generalise. Therefore, the reason may be that semantic based crossovers find the solutions closer to the perfect solution than standard crossover. If so, the question is to what extent can we continuously run GP systems with semantic based crossovers and that are still less overfitting than standard crossover. Future work aims to address this question.

Finally, this thesis proposed a novel way to measure semantics in GP by sampling a number of points from the problem domain. However, other ways to measure semantic could be better. Future research aims to find other ways to qualify semantic in GP to apply to other problem domains.

Appendix A

Semantics Exchanged in Semantic based Crossovers

This chapter analyses the semantics exchanged in two improvement of Semantic Similarity based Crossover (SSC). The two analysed crossovers are Self-Adaptive Successful Execution (SASE) method and the Most Semantic Similarity based Crossover (MSSC). It has been shown in Chapter 7 that the largest proportion of standard crossover (SC) events make a big change in terms of semantics from parents to children and SSC achieves higher locality than SC by increasing the percentage of SSC events that make smaller change. Since SASE and MSSC are two improvements of SSC, it is important to see how the semantics of the subtree exchanged in these crossovers.

Tab. A.1: The percentage of each groups in 7 groups of SASE

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
SASE0	3.80	3.74	4.79	3.02	3.58	3.46	3.48	3.19
SASE1	27.1	27.6	42.4	9.65	17.9	9.27	46.2	42.6
SASE2	25.1	26.2	25.9	29.5	35.2	24.6	19.7	19.8
SASE3	18.9	18.8	10.2	37.6	21.4	21.9	18.9	20.7
SASE4	11.7	10.2	8.82	9.37	6.04	22.6	5.67	6.58
SASE5	9.91	9.86	4.60	5.91	9.21	9.44	3.64	4.35
SASE6	3.48	3.44	3.14	5.00	6.47	8.69	2.37	2.69

Tab. A.2: The percentage of each groups in 7 groups of MSSC

Xovers	F1	F2	F3	F4	F5	F6	F7	F8
MSSC0	1.42	1.51	2.16	1.85	3.15	2.40	0.93	1.08
MSSC1	40.3	40.9	51.3	16.1	22.7	13.3	57.0	50.4
MSSC2	21.1	20.3	20.0	31.7	27.4	25.3	18.2	17.4
MSSC3	14.4	14.6	14.4	30.5	18.3	17.4	20.3	25.8
MSSC4	16.9	16.3	8.90	13.8	11.7	23.3	2.31	3.23
MSSC5	4.77	5.17	1.98	3.69	11.1	13.0	0.87	1.22
MSSC6	0.97	1.17	1.16	2.24	5.42	5.30	0.44	0.69

We conducted an experiment to analyse the semantics exchanged in these crossovers. For SASE we used SASES and for MSSC we set Max_Trial=12. The experimental settings are as follows. Similar to SC and SSC, SASE and MSSC are divided into seven groups (see Chapter 7). These seven groups will be referred to as SASEX and MSSCX (X=0, 1,..., 6). We recored the percentage of SASE and MSSC that lies in each of the seven groups at each generation. The values are averaged over 50 generations and 100 runs. The results are shown in Table A.1 for SASE and Table A.2 for MSSC.

It can be seen from these tables that the portion of SASE and MSSC that makes a small change, in terms of semantics, from parents to children is substantially increased compared to SC. There is about 60% to 90% of SASE lies in SASE1, SASE2 or SASE3, and the similar results with MSSC. At the same time, the percentage of SASE and MSSC that exchanges two semantic equivalent subtrees (SASE0 and MSSC0) and the percentage of SASE and MSSC that swaps two semantic dissimilar subtrees are remarkably reduced. These table provide the evidence that both SASE and MSSC actually achieved their design objective to keep a small change of parents in crossover in terms of semantics.

Bibliography

[1] Evolutionary and neural computation for time series prediction minisite. Website, 2005. `http://tracer.uc3m.es/tws/TimeSeriesWeb/repo.html`.

[2] Cruise Alan. *Meaning in Language: An introduction to Semantics and Pragmatics*. Oxford Textbooks in Linguistics, Cambridge, UK, 2004.

[3] Lee Altenberg. The schema theorem and price's theorem, April 15 1994.

[4] Peter J. Angeline. Adaptive and self-adaptive evolutionary computations. In Marimuthu Palaniswami and Yianni Attikiouzel, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[5] Peter J. Angeline. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 21–29, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[6] Peter J. Angeline. Subtree crossover: Building block engine or macromutation? In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[8] W. Banzhaf. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, CA, 1998.

[9] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[10] Wolfgang Banzhaf, Frank D. Francone, and Peter Nordin. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 300–309, Berlin, Germany, 22-26 September 1996. Springer Verlag.

[11] L. Beadle and C.G. Johnson. Semantically driven crossover in genetic programming. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 111–116. IEEE Press, 2008.

[12] Lawrence Beadle and Colin G Johnson. Semantically driven mutation in genetic programming. In Andy Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press.

[13] Lawrence Beadle and Colin G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, Sep 2009.

[14] Lawrence Charles John Beadle. *Semantic and Structural Analysis of Genetic Programming*. PhD thesis, University of Kent, Canterbury, July 2009.

[15] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, San Francisco, CA, 1995. Morgan Kaufmann.

[16] Jacob West Brian Chess. *Secure Programming with Static Analysis*. Addison-Wesley Professional, 2007.

[17] A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada, January 1981. Computer Science Department, Technical Report TR81-2.

[18] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[19] Edmund K. Burke, Steven Gustafson, Graham Kendall, and Natalio Krasnogor. Is increased diversity in genetic programming beneficial? an analysis of the effects on performance. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1398–1405, Canberra, December 2003. IEEE Press.

[20] Edmund K. Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.

[21] F.Bettens C. Wedekind, T. Seebeck and A.J. Paepke. Mhc-dependent mate preferences in humans. *Proceedings of Biological Sciences*, 260:245–249, 1995.

[22] Mauro Castelli, Luca Manzoni, Sara Silva, and Leonardo Vanneschi. A comparison of the generalization ability of different genetic programming frameworks. In *WCCI 2010 IEEE World Congress on Computational Intelligence*, pages 94–101. IEEE Press, July 2010.

[23] Mauro Castelli, Luca Manzoni, Sara Silva, and Leonardo Vanneschi. A quantitative study of learning and generalization in genetic programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings*

of the 14th European Conference on Genetic Programming, EuroGP 2011, volume 6621 of *LNCS*, pages 25–36, Turin, Italy, 27-29 April 2011. Springer Verlag.

[24] H. Christiansen. A survey of adaptable grammars. *ACM SIGPLAN Notices*, 25(11): 35–44, November 1990.

[25] Robert Cleary. Extending grammatical evolution with attribute grammars: An application to knapsack problems, 2005.

[26] Robert Cleary and Michael O'Neill. An attribute grammar decoder for the 01 multi-constrained knapsack problem. In *Proceedings of the Evolutionary Computation in Combinatorial Optimization*, pages 34–45. Springer Verlag, April 2005.

[27] A. M. Collins and M. R. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior,*, 8:240–247, 1969.

[28] Luis E. Da Costa and Jacques-Andre Landry. Relaxed genetic programming. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 937–938, Seattle, Washington, USA, July 2006. ACM Press.

[29] Dan Costelloe and Conor Ryan. On improving generalisation in genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 61–72, Tuebingen, April 2009. Springer.

[30] Troya Jose Cotta, C. A hybrid genetic algorithm for the 01 multiple knapsack problem. In *Proceeding of the Artificial Neural Nets and Genetic Algorithms*, pages 251–255. Springer-Verlag, l 1998.

[31] P. Cousot. Abstract interpretation: Achievements and perspectives. In *Proceedings of the SSGRR 2000 Computer and Business International Conference*, 2000.

[32] P. Cousot. Abstract interpretation based formal methods and future challenges. *Lecture Notes in Computer Science*, 2000:138–156, 2001.

[33] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms*. Carnegie Mellon University, 1985.

[34] Jason M. Daida, Robert R. Bertram, Stephen A. Stanhope, Jonathan C. Khoo, Shahbaz A. Chaudhary, Omer A. Chaudhri, and John A. Polito II. What makes a problem GP-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165–191, June 2001.

[35] Marina de la Cruz Echeanda, Alfonso Ortega de la Puente, and Manuel Alfonseca. Attribute grammar evolution. In *Proceedings of the IWINAC 2005*, pages 182–191. Springer Verlag Berlin Heidelberg, 2005.

[36] Kalyanmoy Deb. Fitness landscape. In *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[37] Kenneth A. DeJong. *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[38] Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London, 7-11 July 2007. ACM Press. URL `http://doi.acm.org/10.1145/1276958.1277277`.

[39] Orna Grumberg Edmund M., Clarke Jr. and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[40] A. E. Eiben, E. Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 41–50. Springer-Verlag, Sep 2004.

[41] Aniko Ekart and S. Z. Nemeth. A metric for genetic programs and fitness sharing. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller,

Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 259–270, Edinburgh, 15-16 April 2000. Springer-Verlag.

[42] Aniko 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.

[43] MG Elliot and BJ Crespi. Placental invasiveness mediates the evolution of hybrid inviability in mammals. *The American naturalist*, 168(1):114, 2006.

[44] E. A. Emerson. Temporal and modal logic. *Hand book of Theoretical Computer Science*, B:955–1072.

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

[46] Candida Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[47] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, New York, 1966.

[48] Nate Foreman and Matthew Evett. Preventing overfitting in GP with canary functions. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1779–1780, Washington DC, USA, June 2005. ACM Press. ISBN 1-59593-010-8.

[49] Frank D. Francone, Peter Nordin, and Wolfgang Banzhaf. Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 72–80, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[50] R. FriedBerg. A learning machine - part i. *IBM Journal of Research and Development*, 2:2–12, 1958.

[51] R. FriedBerg. A learning machine - part ii. *IBM Journal of Research and Development*, 3:282–287, 1959.

[52] Christian Gagné, Marc Schoenauer, Marc Parizeau, and Marco Tomassini. Genetic programming, validation sets, and parsimony pressure. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Budapest, Hungary, April 2006. Springer. ISBN 3-540-33143-3.

[53] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[54] David E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In *Genetic Algorithms and Simulated Annealing*, pages 74–88. Morgan Kaufmann, 1987.

[55] David E. Goldberg. Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, 3(2):153–171, 1989.

[56] David E. Goldberg and Una-May O'Reilly. Where does the good stuff go, and why? how contextual semantics influence program structure in simple genetic programming. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 16–36, Paris, 14-15 April 1998. Springer-Verlag.

[57] J. Gottlieb and G.R. Raidl. The effects of locality on the dynamics of decoder-based evolutionary search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 283290. ACM, 2000.

[58] John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[59] EVANNAI group. Evolutionary and neural computation for time series prediction minisite. Website, 2005. `http://tracer.uc3m.es/tws/TimeSeriesWeb/repo.html`.

[60] Steven Gustafson, Edmund K. Burke, and Graham Kendall. Sampling of unique structures and behaviours in genetic programming. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 279–288, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[61] Steven Gustafson, Aniko Ekart, Edmund Burke, and Graham Kendall. Problem difficulty and code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 5(3):271–290, September 2004.

[62] Steven Gustafson, Edmund K. Burke, and Natalio Krasnogor. On improving genetic programming for symbolic regression. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 912–919, Edinburgh, UK, 2005. IEEE Press.

[63] Robin Harper and Alan Blair. A self-selecting crossover operator. In Gary G. Yen, Lipo Wang, Piero Bonissone, and Simon M. Lucas, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 5569–5576, Vancouver, 6-21 July 2006. IEEE Press.

[64] S. Hengpraprohm and P. Chongstitvatana. Selective crossover in genetic programming. In *Proceedings of ISCIT International Symposium on Communications and Information Technologies*, pages 14–16, November 2001.

[65] Nguyen Thi Hien and Nguyen Xuan Hoai. A brief overview of population diversity measures in genetic programming. In *Proceedings of 11th Asia-Pacific Workshop on*

Intelligent and Evolutionary Systems, pages 128–139. Vietnamese Military Technical Academy, October 2006.

[66] M. G. Hinchey and J. P. Bowen, editors. *Applications of Formal Methods*. Prentice Hall International Series in Computer Science, 1995. URL `http://www.comlab.ox.ac.uk/archive/formal-methods/afm-book.html`.

[67] Nguyen Xuan Hoai. *A Flexible Presentation for Genetic Programming: Lessons from Natural Language Processing*. PhD thesis, University of New South Wales, 2004.

[68] Nguyen Xuan Hoai, R.I. McKay, and D. Essam. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*, pages 1326–1331. IEEE Press, 2002.

[69] Nguyen Xuan Hoai, Robert I. McKay, and Daryl Essam. Representation and structural difficulty in genetic programming. *IEEE Transection on Evolutionary Computation*, 10(2):157–166, 2006.

[70] Tuan-Hao Hoang, Nguyen Xuan Hoai, Nguyen Thi Hien, R I McKay, and Daryl Essam. Ordertree: a new test problem for genetic programming. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 807–814, Seattle, Washington, USA, 8-12 July 2006. ACM Press. ISBN 1-59593-186-4.

[71] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press, 1972.

[72] Jeffrey Horn and David E. Goldberg. Genetic algorithm difficulty and the modality of fitness landscapes. In L. Darrell Whitley and Michael D. Vose, editors, *FOGA*, pages 243–269. Morgan Kaufmann, 1994. ISBN 1-55860-356-5.

[73] Ting Hu and Wolfgang Banzhaf. The role of population size in rate of evolution in genetic programming. In *Proceedings of the 12th European Conference on Genetic*

Programming, EuroGP 2009, volume 5481 of *LNCS*, pages 85–96, Tuebingen, April 15-17 2009. Springer.

[74] Talib S. Hussain and Roger A. Browse. Attribute grammars for genetic representations of neural networks and syntactic constraints of genetic programming. In *Proceedings of the Workshop on Evolutionary Computation*, June 1998.

[75] Talib S. Hussain and Roger A. Browse. Genetic encoding of neural networks using attribute grammars. In *Proceedings of the CITO Researcher Retreat*, May 1998.

[76] Talib S. Hussain and Roger A. Browse. Genetic operators with dynamic biases that operate on attribute grammar representations of neural networks. In *Proceedings of the Advanced Grammar Techniques Within Genetic Programming and Evolutionary Computation Conference*, pages 83–86, June 1998.

[77] Hitoshi Iba and Takashi Sasaki. Using genetic programming to predict financial data. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 244–251, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[78] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In *Advances in Genetic Programming*, pages 265–284. MIT Press.

[79] Takuya Ito, Hitoshi Iba, and Satoshi Sato. Depth-dependent crossover for genetic programming. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 775–780. IEEE Press, May 1998.

[80] Takuya Ito, Hitoshi Iba, and Satoshi Sato. A self-tuning mechanism for depth-dependent crossover. In *Advances in Genetic Programming*, page 377399. IEEE Press, June 1999.

[81] F. Strozzi E. Gutierrez J.M. Zaldivar, I.M. Galvan and A. Tomasin. Forecasting high waters at venice lagoon using chaotic time series analysis and nonlinear neural networks. *Journal of Hydroinformatics*, 021:61–84, 2000.

[82] Colin Johnson. Deriving genetic programming fitness properties by static analysis. In *Proceedings of the 4th European Conference on Genetic Programming (EuroGP2002)*, pages 299–308. Springer, 2002.

[83] Colin Johnson. Genetic programming with guaranteed constraints. In *Recent Advances in Soft Computing*, pages 134–140. The Nottingham Trent University, 2002.

[84] Colin Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham, 2002.

[85] Colin Johnson. Genetic programming with fitness based on model checking. In *Proceedings of the 10th European Conference on Genetic Programming (EuroGP2002)*, pages 114–124. Springer, 2007.

[86] Colin Johnson. Genetic programming crossover: Does it cross over? In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP2009)*, pages 97–108. Springer, 2009.

[87] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, NM, 1995.

[88] Terry Jones. One operator, one landscape, June 02 1995.

[89] Gal Katz and Doron Peled. Model checking-based genetic programming with an application to mutual exclusion. *Tools and Algorithms for the Construction and Analysis of Systems*, 4963:141–156, 2008.

[90] Gal Katz and Doron Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, 5311:33–47, 2008.

[91] S. A. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *J. Theor. Biol.*, 128:11–45, 1987.

[92] Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Proceedings of EuroGP'2003*, pages 70–82. Springer-Verlag, April 2003.

[93] Back T. Khuri, S. and J. Heitkotter. The zero/one multiple knapsack problem and genetic algorithms. In *Proceeding of the1994 ACM symposium of Applied Computation*, pages 188–193. ACM Press, l 1994.

[94] K. E. Kinnear. A rigorous evaluation of crossover and mutation in genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP*, pages 881–888. IEEE Press, July 1998.

[95] Kenneth E. Kinnear. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[96] Kenneth E. Kinnear. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, volume 2, pages 881–888, San Francisco, USA, 28 March-1 April 1993. IEEE Press.

[97] Donald Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:95, 1968.

[98] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.

[99] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994. ISBN 0-262-11189-6.

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

[101] John R. Koza, Forrest H Bennett III, David Andre, Martin A. Keane, and Scott Brave. *Genetic Programming III Videotape: Human Competitive Machine Intelligence*. Morgan Kaufmann, 340 Pine Street - 6th Floor, San Francisco, CA 94104, USA, 1999.

[102] Krysztof Krawiec and PremysBaw Polewski. Potential fitness for genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 2175–2180. ACM.

[103] Krzysztof Krawiec and Pawel Lichocki. Approximating geometric crossover in semantic space. In Franz Rothlauf, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 987–994. ACM, 2009.

[104] Krzysztof Krawiec and Bartosz Wieloch. Functional modularity for genetic programming. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 995–1002, Montreal, July 2009. ACM.

[105] Ibrahim Kushchu. An evaluation of evolutionary generalisation in genetic programming. *Artificial Intelligence Review*, 18(1):3–14, September 2002.

[106] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transations on Computer Systems*, 5(1):1–11, 1987.

[107] W. B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structure = Automatic Programming!* Kluwer Academic, Boston, 1998.

[108] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1092–1097. Morgan Kaufmann, July 1999.

[109] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In Chris Clack, Kanta Vekaria, and Nadav Zin, editors, *ET'97 Theory and Application of Evolutionary Computation*, pages 59–77, University College London, UK, 15 December 1997.

[110] Jin Li, Zhu Shi, and Xiaoli Li. Genetic programming with wavelet-based indicators for financial forecasting. *Transactions of the Institute of Measurement and Control*, 28(3):285–297, August 2006.

[111] Moshe Looks. On the behavioral diversity of random programs. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1636–1642. ACM Press, 7-11 July 2007.

[112] Sean Luke and Lee Spector. A comparison of crossover and mutation in genetic programming. In *Proceedings of the Second Annual Conference on Genetic Programming*, pages 240–248. San Francisco, CA, USA, April 1997.

[113] Sean Luke and Lee Spector. A revised comparison of crossover and mutation in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 208–213, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[114] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[115] Sébastien Mahler, Denis Robilliard, and Cyril Fonlupt. Tarpeian bloat control and generalization accuracy. In *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, Lausanne, Switzerland, April 2005. Springer.

[116] Hammad Majeed and Conor Ryan. A less destructive, context-aware crossover opera-

tor for GP. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 36–48. Lecture Notes in Computer Science, Springer, April 2006.

[117] Hammad Majeed and Conor Ryan. Context-aware mutation: a modular, context aware mutation operator for genetic programming. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1651–1658, London, 7-11 July 2007. ACM Press.

[118] Keith E. Mathias and L. Darrell Whitley. Genetic operators, the fitness landscape and the traveling salesman problem. In Reinhard Manner and Bernard Manderick, editors, *Parallel Problem Solving from Nature 2, PPSN-II (2nd PPSN'92)*, pages 221–230, Brussels, Belgium, September 1992. Elsevier Science Publishers (Amsterdam).

[119] S. R. Maxwell. Why might some problems be difficult for genetic programming to find solutions? In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, pages 125–128, Stanford University, CA, USA, 28–31 July 1996. Stanford Bookstore.

[120] Ben McKay, Mark J. Willis, and Geoffrey W. Barton. Using a tree structured genetic algorithm to perform symbolic regression. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 487–492, Sheffield, UK, 12-14 September 1995. IEEE.

[121] R. I. (Bob) McKay. An investigation of fitness sharing in genetic programming. *The Australian Journal of Intelligent Information Processing Systems*, 7(1/2):43–51, July 2001. URL `http://sc.snu.ac.kr/PAPERS/AJIIPSfitshr.pdf`.

[122] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3/4):365–396, September 2010.

[123] N.F. McPhee, B. Ohs, and T. Hutchison. Semantic building blocks in genetic programming. In *Proceedings of 11th European Conference on Genetic Programming*, pages 134–145. Springer, 2008.

[124] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 April 2000. Springer-Verlag.

[125] T. Mitchell. *Machine Learning*. McGraw Hill, New York, 1996.

[126] Naoki Mori. A novel diversity measure of genetic programming. In *Randomness and Computation: Joint Workshop "New Horizons in Computing" and "Statistical Mechanical Approach to Probabilistic Information Processing"*, Sendai International Center, Sendai, Japan, 18-21 July 2005. Extended Abstract.

[127] Naoki Mori, Bob McKay, Nguyen Xuan Hoai, Daryl Essam, and Saori Takeuchi. A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 13(3):237–244, 2009.

[128] Pereira F. C. N. and Warren D. H. D. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks, artificial intelligence. In *Proceeding of the Artificial Intelligence*, pages 231–278, 1980.

[129] N.D.Jones and F. Nielson. Abstract interpretation: a semantics based tool for program analysis. *Handbook of Logic in Computer Science*, 1995.

[130] Quang Uy Nguyen, Xuan Hoai Nguyen, and M. O'Neill. Semantic aware crossover for genetic programming: the case for real-valued function regression. In *Proceedings of EuroGP09*, pages 292–302. Springer, April 2009.

[131] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Semantics based mutation in genetic programming: The case for real-valued symbolic regression. In R. Matousek and L. Nolle, editors, *15th International Conference on Soft Computing, Mendel'09*, pages 73–91, Brno, Czech Republic, June 24-26 2009.

[132] Quang Uy Nguyen, Michael O'Neill, Xuan Hoai Nguyen, Bob McKay, and Edgar Galvan Lopez. Semantic similarity based crossover in GP: The case for real-valued function regression. In Pierre Collet, editor, *Evolution Artificielle, 9th International Conference*, Lecture Notes in Computer Science, pages 13–24, October 2009.

[133] Quang Uy Nguyen, Michael O'Neill, Xuan Hoai Nguyen, Bob McKay, and Edgar Galvan Lopez. An analysis of semantic aware crossover. In *Proceedings of the International Symposium on Intelligent Computation and Applications ISICA 2009*. Springer-Verlag, 2009.

[134] Quang Uy Nguyen, Bob McKay, Michael O'Neill, and Xuan Hoai Nguyen. Self-adapting semantic sensitivities for semantic similarity based crossover. In *IEEE Congress on Evolutionary Computation 2010*, Barcelona, Spain, 7-11 July 2010. IEEE Press.

[135] Quang Uy Nguyen, Thi Hien Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Improving the generalisation ability of genetic programming with semantic similarity based crossover. In Anna I. Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, Istanbul, 7-9 April 2010. Springer.

[136] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, and Bob Mckay. Semantics

based crossover for boolean problems. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2010*, Portland, Oregon, 7-11 July 2010. ACM.

[137] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, and Bob McKay. The role of syntactic and semantic locality of crossover in genetic programming. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Guenter Rudolph, editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6239 of *Lecture Notes in Computer Science*, pages 533–542, Krakow, Poland, 11-15 September 2010. Springer.

[138] Quang Uy Nguyen, Michael O'Neill, and Xuan Hoai Nguyen. Predicting the tide with genetic programming and semantic-based crossovers. In *KSE 2010 The Second International Conference on Knowledge and Systems Engineering*, Hanoi, Vietname, 7-9 October 2010. IEEE Computer Society, IEEE Press.

[139] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Examining the landscape of semantic similarity based mutation. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, Dublin, Ireland, 12-16 July 2011. ACM.

[140] Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, R. I. McKay, and Edgar Galvan-Lopez. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 2011. ISSN 1389-2576. doi: doi:10.1007/s10710-010-9121-2.

[141] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 2005.

[142] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer, Springer-Verlag, London, UK, 2007.

[143] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor,

Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, pages 6–22, Tahoe City, California, USA, 9 July 1995.

[144] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, August 2001.

[145] Michael O'Neill and Conor Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In *Proceedings of the Genetic Programming 7th European Conference*, pages 138–149. Springer Verlag, April 2004.

[146] Michael O'Neill, Robert Cleary, and Nikola Nikolov. Solving knapsak problems with attribute grammars. In *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 June 2004.

[147] Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3/4):339–363, September 2010.

[148] Una May O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*, pages 4092–4097. IEEE, October 1997.

[149] Una-May O'Reilly and David E. Goldberg. How fitness structure affects subsolution acquisition in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 269–277, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[150] Una-May O'Reilly and Franz Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel*

Problem Solving from Nature – PPSN III, number 866 in Lecture Notes in Computer Science, pages 397–406, Jerusalem, 9-14 October 1994. Springer-Verlag.

[151] P.A.Whigham. A schema theorem for context-free grammars. Technical report, Department of Computer Science, ADFA, University of New South Wales, Australia, 1994.

[152] P.A.Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, Morgan Kaufmann*, pages 33–41. Morgan Kaufmann, 1995.

[153] P.A.Whigham. Grammatically-based genetic programming. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, pages 178–182. IEEE Press, 1995.

[154] R. Poli and W.B. Langdonand N.F. McPhee. *A Field Guide to Genetic Programming*. http://lulu.com, 2008.

[155] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex, 14-16 April 2003. Springer-Verlag.

[156] Riccardo Poli and W. B. Langdon. Genetic programming with one-point crossover. In *Proceedings of Soft Computing in Engineering Design and Manufacturing Conference*, pages 180–189. Springer-Verlag, June 1997.

[157] Riccardo Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Pro-*

ceedings of the Third Annual Conference, pages 293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[158] Riccardo Poli, Nicholas F. McPhee, and Leonardo Vanneschi. Analysis of the effects of elitism on bloat in linear and tree-based genetic programming. In Rick L. Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, chapter 7, pages 91–111. Springer, Ann Arbor, 15-17May 2008.

[159] Riccardo Poli, Nicholas McPhee, Luca Citi, and Ellery Crane. Memory with memory in tree-based genetic programming. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 25–36, Tuebingen, April 15-17 2009. Springer.

[160] William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 15, pages 299–316. MIT Press, Cambridge, MA, USA, 1996.

[161] Michael J. Cloud Ramon E. Moore, R. Baker Kearfott. *Introduction to interval analysis*. Society for Industrial and Applied Mathematics, 2003.

[162] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[163] C.R. Reeves and J.E. Rowe. *Genetic Algorithms: Principles and Perspectives*. Kluwer Academic Pulisher, 2003.

[164] Jeff Riley and Vic Ciesielski. Fitness landscape analysis for evolutionary non-photorealistic rendering. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.

[165] Daniel Rivero, Juan R. Rabunal, Julian Dorado, and Alejandro Pazos. Time series forecast with anticipation using genetic programming. In Joan Cabestany, Alberto Prieto, and Francisco Sandoval Hernández, editors, *Computational Intelligence and Bioinspired Systems, 8th International Work-Conference on Artificial Neural Networks, IWANN 2005, Proceedings*, volume 3512 of *Lecture Notes in Computer Science*, pages 968–975, Vilanova i la Geltrú, Barcelona, Spain,, June 8-10 2005. Springer.

[166] Emilio Del Rosal, Alfonso Ortega, Manuel Alfonseca, and Marina De La Cruz. Christiansen grammar evolution: grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):"", Fubruary 2007.

[167] Justinian P. Rosca. Entropy-driven adaptive representation. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, Tahoe City, California, USA, 9 July 1995.

[168] Justinian P. Rosca and Dana H. Ballard. Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, Feb 1994.

[169] Franz Rothlauf and David Goldberg. Redundant Representations in Evolutionary Algorithms. *Evolutionary Computation*, 11(4):381–415, 2003.

[170] Franz Rothlauf and Marie Oetzel. On the locality of grammatical evolution. Working Paper 11/2005, Department of Business Administration and Information Systems, University of Mannheim, D-68131 Mannheim, Germany, December 2005.

[171] Franz Rothlauf and Marie Oetzel. On the locality of grammatical evolution. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 320–330. Lecture Notes in Computer Science, Springer, April 2006.

[172] Massimo Santini and Andrea Tettamanzi. Genetic programming for financial time series prediction. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan,

Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 361–370, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[173] J. N. Shutt. Recursive adaptable grammars, August 1993.

[174] Detlef Sieling and Ingo Wegener. Reduction of obdds in linear time. *Information Processing Letters*, 48(3):139–144, November 1993.

[175] Sara Silva and Stephen Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 159–170, Tuebingen, April 2009. Springer.

[176] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[177] Terence Soule and James A. Foster. Code size and depth flows in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[178] Terence Soule and Robert B. Heckendorn. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(3):283–309, September 2002.

[179] W. A. Tackett and A. Carmi. The unique implications of brood selection for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[180] Walter Alden Tackett. *Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, USA, 1994.

[181] L. Vanneschi, M. Tomassini, P. Collard, and M. Clergue. Fitness distance correlation in genetic programming: A constructive counterexample. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 289–296, Canberra, 2003. IEEE Press.

[182] Leonardo Vanneschi and Steven Gustafson. Using crossover based similarity measure to improve genetic programming generalization ability. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1139–1146, Montreal, 8-12 July 2009. ACM.

[183] Leonardo Vanneschi, Marco Tomassini, Manuel Clergue, and Philippe Collard. Difficulty of unimodal and multimodal landscapes in genetic programming. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1788–1799, Chicago, 12-16 July 2003. Springer-Verlag. ISBN 3-540-40603-4.

[184] Leonardo Vanneschi, Marco Tomassini, Philippe Collard, and Manuel Clergue. Fitness distance correlation in structural mutation genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 455–464, Essex, 14-16 April 2003. Springer-Verlag.

[185] Leonardo Vanneschi, Marco Tomassini, Philippe Collard, and Manuel Clergue. A survey of problem difficulty in genetic programming. In Stefania Bandini and Sara Manzoni, editors, *AI*IA 2005: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence, Proceedings*, volume 3673 of *Lecture Notes in Computer Science*, pages 66–77, Milan, Italy, September 21-23 2005. Springer.

[186] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 877–884, Portland, Oregon, USA, 7-11 July 2010. ACM. ISBN 978-1-4503-0072-8. doi: doi:10.1145/1830483.1830643.

[187] Vesselin K. Vassilev, Julian F. Miller, and Terence C. Fogarty. Digital circuit evolution and fitness landscapes. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1299–1306, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[188] Vesselin K. Vassilev, Terence C. Fogarty, and Julian F. Miller. Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60, Spring 2000.

[189] Andrew H. Watson and Ian C. Parmee. Steady state genetic programming with constrained complexity crossover using species sub population. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 315–321, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.

[190] Edward D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[191] P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 July 1995.

[192] David R. White and Simon Poulding. A rigorous evaluation of crossover and mutation in genetic programming. In Leonardo Vanneschi, Steven Gustafson, Alberto

Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 220–231, Tuebingen, April 15-17 2009. Springer.

[193] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation. In James D. Schaffer, editor, *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 116–121, San Mateo, CA, 1989. Morgan Kaufmann.

[194] Man Leung Wong and Kwong Sak Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, 1995.

[195] Man Leung Wong and Kwong Sak Leung. Learning programs in different paradigms using genetic programming. In *Proceedings of the Fourth Congress of the Italian Association for Artificial Intelligence*. Springer-Verlag, 1995.

[196] Man Leung Wong and Kwong Sak Leung. Combining genetic programming and inductive logic programming using logic grammars. In *1995 IEEE Conference on Evolutionary Computation*, pages 733–736. IEEE Press, November 1995.

[197] Man Leung Wong and Kwong Sak Leung. Applying logic grammars to induce subfunctions in genetic programming. In *1995 IEEE Conference on Evolutionary Computation*, pages 737–740. IEEE Press, November 1995.

[198] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Kluwer Academic Publishers, January 2000.

[199] S. Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proceedings of the Sixth Congress on Genetics*, volume 1, page 365, 1932.