MINISTRY OF EDUCATION AND TRAINING

LE QUY DON TECHNICAL UNIVERSITY

CHU THI HUONG

SEMANTICS-BASED SELECTION AND CODE BLOAT REDUCTION TECHNIQUES FOR GENETIC PROGRAMMIN

DOCTORAL DISSERTATION: MATHEMATICAL FOUNDATION FOR INFORMATICS

HA NOI - 2020

MINISTRY OF EDUCATION AND TRAINING
LE QUY DON TECHNICAL UNIVERSITY

CHU THI HUONG

SEMANTICS-BASED SELECTION AND CODE BLOAT REDUCTION TECHNIQUES FOR GENETIC PROGRAMMIN

DOCTORAL DISSERTATION

Major: Mathematical Foundations for Informatics
Code: 946 0110

RESEARCH SUPERVISORS:
1. **Assoc. Prof.Dr. Nguyen Quang Uy**
2. **Assoc. Prof. Dr. Nguyen Xuan Hoai**

HA NOI - 2020

ASSURANCE

I certify that this dissertation is a research work done by the author under the guidance of the research supervisors. The dissertation has used citation information from many different references, and the citation information is clearly stated. Experimental results presented in the dissertation are completely honest and not published by any other author or work.

Author

Chu Thi Huong

ACKNOWLEDGEMENTS

The first person I would like to thank is my supervisor, Dr Nguyen Quang Uy, the lecturer of Faculty of Information Technology, Le Quy Don Technical University, for directly guiding me through the PhD progress. Dr Uy's enthusiasm is the power source to motivate me to carry out this research. His guide has inspired much of the research in this dissertation.

I also wish to thank my co-supervisor, Assoc. Prof. Dr Nguyen Xuan Hoai at AI Academy. He has given and discussed a lot of new issues with me. Working with Prof Hoai, I have learnt how to do research systematically. Particularly, I would like to thank the leaders and lecturers of the Faculty of Information Technology, Le Quy Don Technical University for supporting me with favorable conditions and cheerfully helping me in the study and research process.

Last, but most important, I also would like to thank my family, my parents for always encouraging me, especially my husband, Nguyen Cong Minh for sharing a lot of happiness and difficulty in the life with me, my children, Nguyen Cong Hung and Nguyen Minh Hang for trying to grow up and study by themselves.

Author

Chu Thi Huong

CONTENTS

Contents ....

Abbreviations v

List of figures vii

List of tables ix

INTRODUCTION 1

Chapter 1. BACKGROUNDS 7

1.1. Genetic Programming 7

1.1.1. GP Algorithm 8

1.1.2. Representation of Candidate Solutions 9

1.1.3. Initialising the Population 10

1.1.4. Fitness Evaluation 11

1.1.5. GP Selection 12

1.1.6. Genetic Operators 14

1.1.7. GP parameters 16

1.1.8. GP benchmark problems 18

1.2. Some Variants of GP 18

1.2.1. Linear Genetic Programming 20

1.2.2. Cartesian Genetic Programming 21

1.2.3. Multiple Subpopulations GP 21

1.3. Semantics in GP 23

1.3.1. GP Semantics 23

1.3.2. Survey of semantic methods in GP 27
1.3.3. Semantics in selection and control of code bloat 35
1.4. Semantic Backpropagation 37
1.5. Statistical Hypothesis Test 38
1.6. Conclusion 40

Chapter 2. TOURNAMENT SELECTION USING SEMANTICS 41

2.1. Introduction 41
2.2. Tournament Selection Strategies 43
2.2.1. Sampling strategies 44
2.2.2. Selecting strategies 45
2.3. Tournament Selection based on Semantics 48
2.3.1. Statistics Tournament Selection with Random 49
2.3.2. Statistics Tournament Selection with Size 50
2.3.3. Statistics Tournament Selection with Probability 51
2.4. Experimental Settings 54
2.4.1. Symbolic Regression Problems 54
2.4.2. Parameter Settings 54
2.5. Results and Discussions 57
2.5.1. Performance Analysis of Statistics Tournament Selection 58
2.5.2. Combining Semantic Tournament Selection with Semantic Crossover 65
2.5.3. Performance Analysis on The Noisy Data 69
2.6. Conclusion 77

Chapter 3. SEMANTIC APPROXIMATION FOR REDUCING CODE BLOAT **78**

3.1. Introduction 78

3.2. Controlling GP Code Bloat 81

3.2.1. Constraining Individual Size 81

3.2.2. Adjusting Selection Techniques 81

3.2.3. Designing Genetic Operators 83

3.3. Methods 85

3.3.1. Semantic Approximation 85

3.3.2. Subtree Approximation 87

3.3.3. Desired Approximation 89

3.4. Experimental Settings 90

3.5. Performance Analysis 92

3.5.1. Training Error 92

3.5.2. Generalization Ability 96

3.5.3. Solution Size 98

3.5.4. Computational Time 99

3.6. Bloat, Overfitting and Complexity Analysis 102

3.6.1. Bloat Analysis 102

3.6.2. Overfitting Analysis 103

3.6.3. Function Complexity Analysis 107

3.7. Comparing with Machine Learning Algorithms 109

3.8. Applying semantic methods for time series forecasting 110

3.8.1. Some other versions 112

3.8.2. Time series prediction model and parameter settings ... 113

3.8.3. Results and Discussion 115

3.9. Conclusion 123

CONCLUSIONS AND FUTURE WORK 125

PUBLICATIONS 129

BIBLIOGRAPHY 131

Appendix 146

ABBREVIATIONS

Abbreviation	Meaning
AGSX	Angle-aware Geometric Semantic Crossover
BMOPP	Biased Multi-Objective Parsimony Pressure method
CGP	Cartesian Genetic Programming
CM	Competent Mutation
CTS	Competent Tournament Selection
CX	Competent Crossover
DA	Desired Approximation
EA	Evolutionary Algorithm
Flat-OE	Flat Target Distribution
GA	Genetic Algorithms
GCSC	Guaranteed Change Semantic Crossover
GP	Genetic Programming
GSGP	Geometric Semantic Genetic Programming
GSGP-Red	GSGP with Reduced trees
KLX	Krawiec and Lichocki Geometric Crossover
LCSC	Locality Controlled Semantic Crossover
LGP	Linear Genetic Programming
LGX	Locally Geometric Semantic Crossover
LPP	Lexicographic Parsimony Pressure
MODO	Multi-Objective Desired Operator
MORSM	Multi-Objective Randomized Similarity Mutation
MS-GP	Multiple Subpopulations GP
MSSC	Most Semantically Similar Crossover

Abbreviation	**Meaning**
OE	Operator Equalisation
PC	Perpendicular Crossover
PP	Prune and Plant
PP-AT	Prune and Plant based on Approximate Terminal
RCL	Restricted Candidate List
RDO	Random Desired Operator
ROBDDs	Reduced Ordered Binary Decision Diagrams
RSM	Random Segment Mutation
SA	Subtree Approximation
SAC	Semantics Aware Crossover
SAS-GP	Substituting a subtree with an Approximate Subprogram
SAT	Semantic Approximation Technique
SAT-GP	Substituting a subtree with an Approximate Terminal
SDC	Semantically-Driven Crossover
SiS	Semantic in Selection
SSC	Semantic Similarity based Crossover
SS+LPE	Spatial Structure with Lexicographic Parsimonious Elitism
TS-P	Statistics Tournament Selection with Probability
TS-R	Statistics Tournament Selection with Random
TS-S	Statistics Tournament Selection with Size

LIST OF FIGURES

1 Number of articles about GP 2

2 Number of articles using semantics in GP 4

1.1 GP syntax tree representing $max(x + x, x + 3 * y)$. 9

1.2 An example of crossover operator. 14

1.3 An example of mutation operator. 15

1.4 An example of LGP program. 20

1.5 An example of CGP program. 21

1.6 Structure of MS-GP. 22

1.7 Running the program p on all fitness cases 25

1.8 An example of calculating the desired semantics of the selected node N. 37

2.1 Testing error and Population size over the generations with tour-size=3. 72

3.1 An example of Semantic Approximation 86

3.2 (a) the original tree with the selected subtree, (b) the small generated tree, and (c) the new tree obtained by substituting a branch of tree (a) with an approximate tree grown from the small tree (b). 88

3.3 Average bloat over generations on four problems F1, F13, F17 and F25. 104

3.4 Average overfitting over the generations on four problems F1, F13, F17 and F25. 106

3.5 Average complexity of the best individual over the generations on four problems F1, F13, F17 and F25. 108

3.6 An example of PP-AT. 113
3.7 Plot of log(unit sale + 1) from 9/1/2016 to 12/31/2016. . . . 114
3.8 Testing error over the generations. 119
3.9 Average size of population over the generations. 121

LIST OF TABLES

1.1 Summary of Evolutionary Parameter Values 17

1.2 GP benchmark regression problems. Variable names are, in order, x, y, z, v and w. Several benchmark problems intentionally omit variables from the function. In the training and testing sets, $U[a, b]$ is uniform random samples drawn from a to b inclusive, and $E[a, b]$ is a grid of points evenly spaced from a to b inclusive 19

2.1 Problems for testing statistics tournament selection techniques 55

2.2 Evolutionary Parameter Values. 56

2.3 Mean of best fitness with tour-size=3 (the left) and tour-size=7 (the right). 59

2.4 Median of testing error with tour-size=3 (the left) and tour-size=7 (the right) 60

2.5 Average of solution's size with tour-size=3 (the left) and tour-size=7 (the right) 62

2.6 Average semantic distance with tour size=3. Bold indicates the value of SiS and TS-S is greater than the value of GP. 63

2.7 Average percentage of rejecting the null hypothesis in Wilcoxon test of TS-R and TS-S with tour-size=3. 64

2.8 Median of testing error of TS-RDO and four other techniques with tour-size=3 (the left) and tour-size=7 (the right) 67

2.9 Average of solutions size of TS-RDO and four other techniques with tour-size=3 (the left) and tour-size=7 (the right) 68
2.10 Median of testing error on the noisy data with tour-size=3 (the left) and tour-size=7 (the right) 70
2.11 Average running time in seconds on noisy data with tour-size=3 (the left) and tour-size=7 (the right) 74
2.12 Average execution time of a run (shorted as Run) and average execution time of selection step (shorted as Tour) of GP and TS-S in seconds on noisy data with tour size=3. . 75
2.13 Median of testing error and average running time in seconds on noisy data with tour-size=3 when the statistical test is conducted on 100 fitness cases. The left is the median of the testing error and the right is the average running time. 76

3.1 Evolutionary parameter values 91
3.2 Mean of the best fitness 93
3.3 Average percentage of better offspring 95
3.4 Median of testing error 97
3.5 Average size of solutions 100
3.6 Average running time in seconds 101
3.7 Values of the grid search for SVR, DT and RF 109
3.8 Comparison of the testing error of GP and machine learning systems. The best results are underlined. 111
3.9 Mean of the best fitness 116
3.10 Median of testing errors 117
3.11 Average of solution's size 120
3.12 Average running time in seconds 122
A.1 Mean best fitness on training noise data with tour-size=3 (the left) and tour-size=7 (the right) 147

A.2 Average of solutions size on training noise data with tour-size=3 (the left) and tour-size=7 (the right) 148
A.3 Mean of best fitness with tour size=5. The left is original data and the right is noise data. 149
A.4 Median of testing error with tour size=5. The left is original data and the right is noise data. 150
A.5 Average of solution's size with tour size=5. The left is original data and the right is noise data. 151
A.6 Mean of best fitness of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data. 152
A.7 Median of fittest of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data. 153
A.8 Average of solutions size of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data. 154

INTRODUCTION

Machine learning is a branch of artificial intelligence that provides the capability to automatically learn and improve from past experience to make future decisions. The fundamental goal of machine learning is to generalize or induce an unknown rule from examples of the rule's application. Machine learning has been studied and applied in many different fields of science and technology. It can be said that most smart systems today are the application of one or more machine learning methods.

Genetic Programming (GP) is considered as a machine learning method that allows computer programs encoded as a set of tree structures to be evolved using an evolutionary algorithm [50, 97]. A GP system is started by initializing a population of individuals. The population is then evolved for a number of generations using genetic operators such as crossover and mutation. At each generation, the individuals are evaluated using a fitness function, and a selection schema is used to choose better individuals to create the next population. The evolutionary process is continued until a desired solution is found or when the maximum number of generations is reached.

Since first introduced in the 1990s, GP has been successfully applied in a wide range of problems, especially with applications in classification, control and regression. Figure 1 surveys the number of GP articles indexed in Scopus[1] over a period of 19 years, from 2000 to 2018. The figure shows that GP studies have rapidly increased in the 2010s, about 750 articles per year, and remained roughly stable to date.

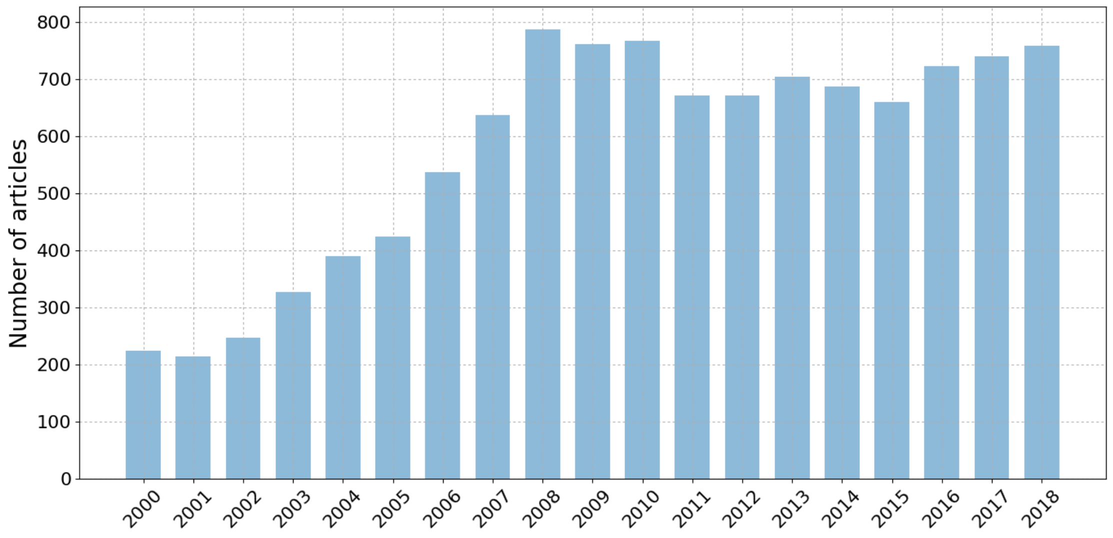


Figure 1: Number of articles about GP

Comparing to other machine learning methods, GP has some advantages. Firstly, GP has the ability to simultaneously learn models (the structure of solutions) and parameters of the models while other methods often have to pre-define models and then find parameters. Secondly, the solutions found by GP are probably interpretable. Recently, several researches have shown that GP can be used to evolve both the architecture and the weights of a Deep Learning model effectively [40, 109]. Interestingly, in 2018, GP outperformed neural networks and deep learning machines at video games [46, 47, 121][2].

However, despite such advantages, GP is not well known in the mainstream AI and Machine Learning communities. One of the main rea-

[1]https://db.vista.gov.vn:2088/search/form.uri?display=basic

[2]https://www.technologyreview.com/s/611568/evolutionary-algorithm-outperforms-deep-learning-machines-at-video-games/

sons is that the evolutionary process is often guided by only syntactic aspects of GP representation. Consequently, there is complex, rugged genotype-phenotype mapping, and low similarity of offspring to parents. An offspring generated by changing syntax may not produce the desired result, or a small change in syntax can significantly change its output (behavior). For example, if we replace the structure $x * 0.001$ in a tree with the structure $x/0.001$ that is a small structural change (replacing '*' with '/') but leads to a significant change in behavior. Algorithms based solely on the structure as that often do not achieve high efficiency since, from a programmer's perspective, programs must be correct not only syntactically, but also semantically. Thus, incorporating semantic awareness in the GP evolutionary process could potentially improve performance and extend the applicability to problems that are difficult to deal with using purely syntactic approaches.

The idea of incorporating semantics into GP evolutionary process is not entirely new. To enhance GP performance, several methods incorporated semantic information have been proposed. Such approaches often either modify operators, design new operators or promote locality and semantic diversity. Figure 2 gives information about the number of articles using semantics published on Scopus. It is clear that the number of studies using semantics in GP has increased rapidly in recent years. However, there are few researches of using semantics for selection and bloat control. In GP, selection mechanism plays a very important role in the performance of GP while standard selection methods only use fitness values and ignore other finer-grained information such as seman-

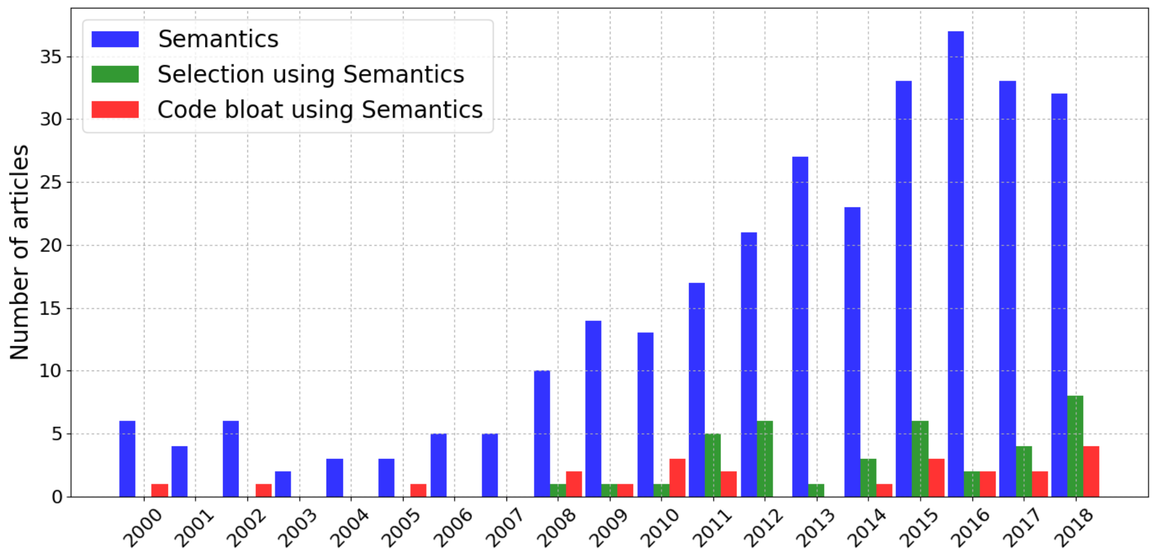


Figure 2: Number of articles using semantics in GP

tics. Hence, we hypothesize that integrating semantic information into selection methods probably promotes semantic diversity and improves GP performance.

Moreover, code bloat is a well-known phenomenon in GP and is one of the GP major issues that many researchers have attempted to address. These studies used a variety of strategies such as setting the maximum depth for the evolved trees, punishing the largest individuals, or adjusting population size distribution at each generation. However, the bloat control methods are often difficult to fit the training data leading to a reduction in GP performance. It is reckoned that under the guidance of semantic information, the bloat control methods will achieve better performance.

From that, this dissertation focuses on two main objectives, including improving selection performance and overcoming code bloat phenomenon in GP. In order to achieve these objectives, the dissertation uses an approach that combines theoretical analysis with experiment, and utilizes techniques in the fields of statistics, formal semantics, machine leaning and optimization to improve the performance of GP. The objectives have been completed by proposing several novel methods. To evaluate the effectiveness of the proposed methods and compare them with the best and new methods in the same field, we tested these methods on benchmarking problems and datasets taken from UCI database. These methods have improved GP performance, promoted semantic diversity and reduced GP code bloat. The main contributions of the dissertation are outlined as follows.

- Three new semantics based tournament selection methods are proposed. A novel comparison between individuals based on a statistical analysis of their semantics is introduced. From that, three variants of the selection strategy are proposed. These methods promote semantic diversity and reduce code bloat in GP.
- A semantic approximation technique is proposed. We propose a new technique that allows to grow a small (sub)tree with the semantics approximate to a given target semantics.
- New bloat control methods based on semantic approximation are introduced. Inspired by the semantic approximation technique, a number of methods for reducing GP code bloat are proposed and evaluated on a large set of regression problems and a real-world time series forecasting.

Additionally, we also propose a new variant without losing the basic structure of GP.

The results of the dissertation include 7 papers, of which 5 papers (1 SCI-Q1 journal paper, 3 International Conference papers and 1 domestic scientific journal paper) were published, and 2 papers (1 SCIE-Q1 journal paper and 1 domestic scientific journal paper) are under review.

The dissertation includes three chapters, introduction, conclusion and future work. The remainder of the dissertation is organised as follows. Chapter 1 gives a more detailed introduction to GP and a brief introduction to some variants of GP, including our new proposed variant. Semantic concepts and a survey of different ways of using semantics in GP are also included. Finally, this chapter introduces a semantic backpropagation algorithm and a statistical hypothesis test.

Chapter 2 presents the proposed forms of tournament selection. We introduce a novel comparison between individuals by using a statistical analysis of GP error vectors. Based on that, some variants of tournament selection are proposed to promote semantic diversity and to explore the potential of the approach to control program bloat. We evaluate these methods on a large number of regression problems and their noisy variants. Experimental results are discussed and evaluated carefully.

Chapter 3 introduces a new proposed technique to grow a (sub)tree that approximates to a target semantic vector. Using this technique, several methods for reducing code bloat are introduced. The proposed methods are extensively evaluated on a large set of regression problems and a real-world time series forecasting problem. The results and discussions are presented in detail.

Chapter 1

BACKGROUNDS

This chapter introduces Genetic Programming (GP) including the algorithm and the basic components of the algorithm. After that, some variants of GP including our proposed variant of GP structure [C4] are briefly shown. The concepts related to semantics in GP are then presented. Next, the chapter reviews the previous semantic methods. Finally, a semantic backpropagation algorithm and a statistical hypothesis test are introduced.

1.1. Genetic Programming

Genetic Programming (GP) is an Evolutionary Algorithm (EA) that automatically finds the solutions of unknown structure for a problem [50, 97]. It is also considered as a metaheuristic-based machine learning method which finds solutions in form of computer programs for a given problem through an evolutionary process. GP has been successfully used as an automatic programming tool, a machine learning tool and an automatic problem-solving engine [97]. Several applications of GP are curve fitting, data modeling, symbolic regression, feature selection and classification. GP applications are applied to a number of problems in many fields such as pattern recognition, software engineering, computer vision, medical and cyber security [30, 57, 61, 62, 118, 128]. Particularly, there are some typical results that are competitive with human-produced results presented in the annual "Humies" competition at "Genetic and Evolutionary Computation Conference (GECCO)"[1]. The algorithm of GP and its components are thoroughly introduced in the next subsection.

1.1.1. GP Algorithm

Technically, GP is considered as an evolutionary algorithm, so it shares a number of common characteristics with other EAs. The GP algorithm and its basic steps are shown in Algorithm 1 [97].

Algorithm 1: GP algorithm

```
Input: The training set of fitness cases.
Output: a solution of the problem.
1. Randomly create an initial population of programs from the available
   primitives;
repeat
    2. Execute each program and evaluate its fitness;
    3. Select one or two program(s) from the population with a probability
       based on fitness to participate in genetic operators;
    4. Create new individual program(s) by applying genetic operators with
       specified probabilities;
until an acceptable solution is found or some other stopping condition is met;
return the best-so-far individual;
```

The first step in running a GP system is to create an initial population of computer programs. Normally, this population is randomly generated with respect to some constraints in terms of syntax and used as a starting point of a GP algorithm. GP then finds out how well a program works by running it, and then comparing its behaviour to some objectives of

[1]http://www.human-competitive.org/awards

the problem (step 2). This comparison is quantified to give a numeric value called fitness. Those programs that do well are chosen to breed (step 3) and produce new programs for the next generation (step 4). Generation by generation, GP transforms populations of programs into new, hopefully better, populations of programs by repeating steps 2-4 until a termination condition is met. Commonly, the termination condition is met when either a acceptable solution is found, or the maximum number of evolutionary generations is reached.

1.1.2. Representation of Candidate Solutions

For evolutionary algorithms, practitioners need to select an appropriate representation of candidate solutions before they use them to solve a problem. In GP, there are several representation formats that can be used such as tree-based representation [50, 81, 97], linear representation [87], grammar-based representation [66] and graph-based representation [69]. Tree-based representation is the most popular form in GP. Thus, this dissertation mainly focuses on tree-based representation.

A tree-based representation of an individual in a GP population is

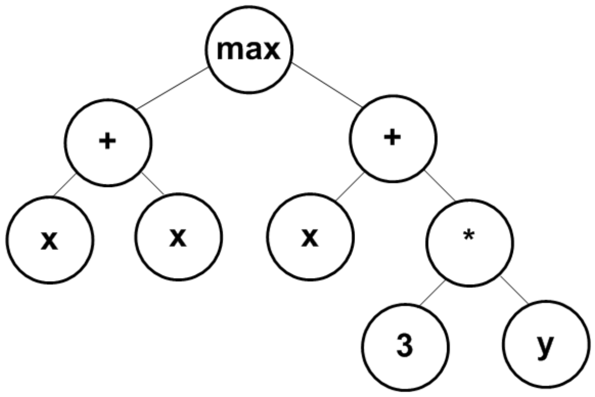


Figure 1.1: GP syntax tree representing $max(x + x, x + 3 * y)$.

constructed from two primitive sets: a function set and a terminal set. The internal nodes of the tree (including the root node) are taken from the function set, and the leaf nodes are taken from the terminal set. Figure 1.1 shows the tree representation of the computer program `max(x+x,x+3*y)`. The variables and constants in the program (`x,y` and `3`) are the leaves of the tree, while the arithmetic operations (`+,*` and `max`) are internal nodes.

The function set, $\mathbb{F} = \{f_1, f_2, ..., f_n\}$, used in GP is typically driven by the nature of the problem domain. It might include arithmetic functions (`+,-,*,/`), mathematical functions (`sin, cos, exp`), boolean functions (`and, or, not`), conditional functions (such as `if-then-else`), and other functions that can be defined to fit the specific problem. Each function in $\mathbb{F}$ has a fixed number of arguments that is considered as its arity. For instance, function `+` has 2-arity, and function `sin` has 1-arity.

The terminal set, $\mathbb{T} = \{t_1, t_2, ..., t_m\}$, that often consists of a number input variables and several constants is comprised of the inputs to a GP program. All terminals have an arity of zero. Similarly to any other machine learning systems, input variables are the features of the observations of a given problem, so these inputs becomes part of GP training and test sets.

1.1.3. Initialising the Population

To start a GP evolutionary process, individuals in an initial population are typically randomly generated. For a tree-based GP system, there are three popular approaches to generate an initial population, including the *grow* method, the *full* method and the *ramped half-and-half* method [50, 97].

The grow method starts to generate an individual by randomly selecting a function f_i in $\mathbb{F}$. 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 $\mathbb{F} \cup \mathbb{T}$ as the children of the root node. If a terminal is chosen, this branch of the tree is terminated. Otherwise, a function is selected, the process is recursively applied for that function. The maximal depth of a tree is usually used to limit the size of an initial individual. If the depth of the tree reaches the maximal depth, the children nodes are only selected in $\mathbb{T}$.

The full method constructs a tree by selecting only functions from $\mathbb{F}$ until the tree reaches the maximal depth. At this depth, only primitives from $\mathbb{T}$ are chosen.

The ramped half-and-half method is the most popular method that is widely used by GP practitioners nowadays. The ramped half-and-half initialisation of the population is in fact the combination of the grow and full initialisation, in which half of the population is created by the grow method, and the remaining half is generated by the full method. The method is done using a range of depth limits to help ensure that we generate trees having a variety of sizes and shapes.

1.1.4. Fitness Evaluation

Each individual in the population is assigned a numerical value called fitness. The fitness of an individual demonstrates its ability to solve the problem. Fitness can be measured in many different ways. For example, it may be the amount of *error* between its output and the desired output, the amount of *time* required to bring a system to a desired target state or the *accuracy* of a program in recognising patterns or classifying objects. Fitness that directly reflects the ability of an individual to solve the problem as above is also called *raw* fitness. In many situations, raw fitness can be standardised (it is called *standardised* fitness) to provide some conveniences.

Fitness is often evaluated based on the training set of a problem that is called fitness cases. For example, let a set of n pairs (input-output) values, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, be the fitness cases of a regression symbolic problem, then the fitness of an individual i can be defined as Mean Square Error ($f_{MSE}(i)$ in Equation 1.1) or Mean Absolute Error ($f_{MAE}(i)$ in Equation 1.2).

$$f_{MSE}(i) = \frac{1}{n}\sum_{j=1}^{n}(\widehat{y_j} - y_j)^2 \tag{1.1}$$

$$f_{MAE}(i) = \frac{1}{n}\sum_{j=1}^{n}|\widehat{y_j} - y_j| \tag{1.2}$$

where $\widehat{y_j}$ is the output of the individual i with the input is x_j.

1.1.5. GP Selection

The selection mechanism plays a very important role in the performance of GP. Selection is a process whereby certain individuals are selected from the current generation that would serve as parents for the next generation. The individuals are selected probabilistically such that the better performing individuals have a higher chance of getting selected. The most commonly used selection method in GP is Tournament selection [26].

Algorithm 2: Tournament Selection

```
Input: Tournament size: $TourSize$.
Output: The winner individual.
$A \longleftarrow$ Randomly select a individual from the population;
for $i \leftarrow 1$ to $TourSize$ do
    $B \longleftarrow$ Randomly select a individual from the population;
    if $B$ is better than $A$ then
        $A \longleftarrow B$;
    end
end
The winner individual $\longleftarrow$A ;
```

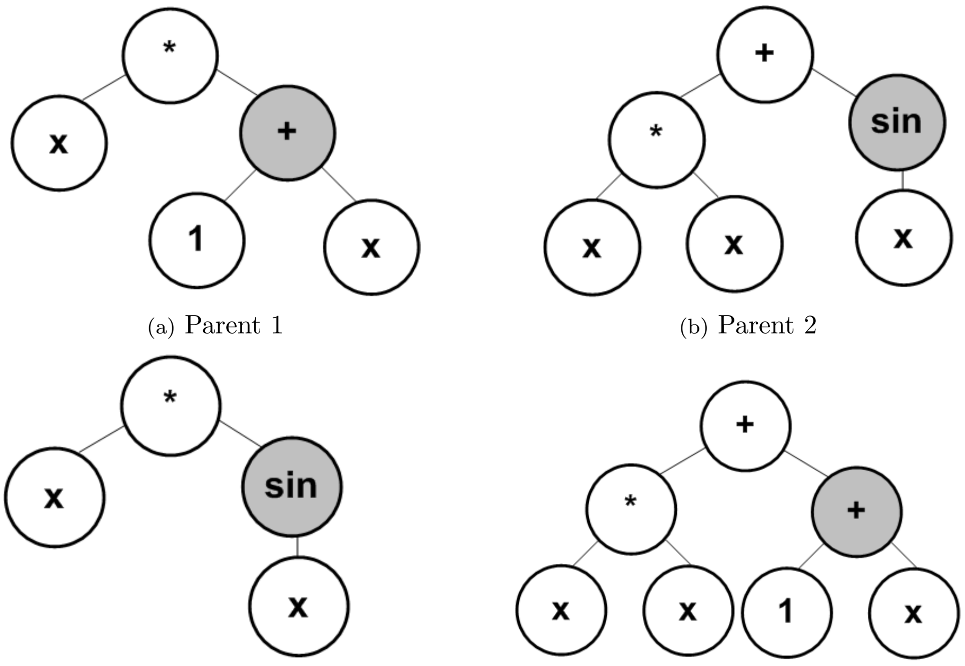
Algorithm 2 presents the algorithm of standard tournament selection. In tournament selection, a number of individuals (tournament size) are randomly selected from the population. These individuals are compared with each other, and the winner (in terms of better fitness) is selected to be the parent. This process is then repeated n times where n is the population size. Note that tournament selection only looks at which an individual is better than another. It does not require a comparison of the fitness between all individuals and need to know how much better. This may help to save a large amount of processing time and provide an easy way to parallelize the algorithms. Tournament selection based methods are reviewed in Section 2.2.

Additionally, there are some other popular selection mechanisms, such as Fitness Proportionate Selection and Ranking Selection [10]. In Fitness Proportionate Selection, every individual can become a parent with a probability which is proportional to its fitness. Differently, Ranking Selection removes the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred to the lower ranked ones.

1.1.6. Genetic Operators

Like in most other EAs, individuals in the new population are formed by using genetic operators. There are three main operators, including crossover, mutation and reproduction.

Crossover operator uses two individuals selected from the current generation through the selection process to produce two different indi-

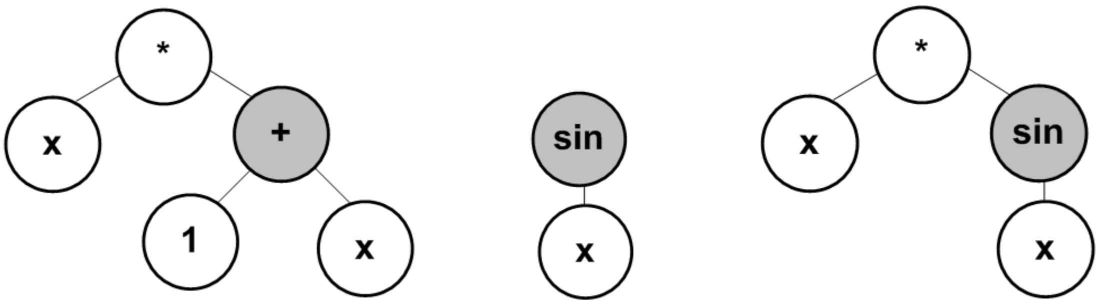


(a) Parent 1 (b) Parent 2 (c) Child 1 (d) Child 2

Figure 1.2: An example of crossover operator.

viduals for the next generation. In this technique, genetic material from two individuals is mixed to form offspring. In the tree-based representation, two parents are selected by a selection method, then one subtree is randomly selected in each parent. The crossover is executed by simply swapping the two chosen subtrees, and the resultant offspring are added to the next generation. Figure 1.2 describes the crossover process.

Mutation operator becomes an important operator which provides diversity to the population. It is an asexual operator with only one parent. Figure 1.3 illustrates the mutation process. The mutation operator selects a random mutation node in a parent tree and replaces this subtree at that node by a newly generated subtree.



(a) Parent (b) A randomly generated subtree (c) Child

Figure 1.3: An example of mutation operator.

Reproduction operator, the other main operator, is performed by simply copying a selected individual from the current generation to the next generation.

Practically, probabilities are used to select which of operators should be utilized to generate an offspring. In GP, operators are normally mutually exclusive [97]. Typically, crossover is applied with the highest probability, often around 0.9. Conversely, mutation is used a much smaller probability, about around 0.1. When the probabilistic choice of crossover and mutation add up to a value p, reproduction is also used, with a probability of $1 - p$.

1.1.7. GP parameters

To begin a GP system, users must specify a number of control parameters. The decisions are critically important as they have a limiting effect on the search space of possible programs. Too great a limit may remove all chance of evolving an acceptable individual. However, not restricting the search-space sufficiently has its own issues.

The most important control parameter is the population size. A larger population allows for a greater exploration of the problem space at each generation and increases the chance of evolving a solution. In general, the more complex a problem is, the greater population size is needed. Other control parameters include the probabilities of performing the genetic operators, the maximum size for programs, the maximum number of generations and other details of the run. 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 the experience of GP practitioners. The following list shows the common parameters that GP practitioners often specify before running GP.

- 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.

In previous researches, the researchers have selected several GP parameter sets to investigate the effectiveness of algorithms. Koza [49] recommended setting values for GP parameters: The population size is 500, the maximum number of generations is 50, the selection method is Fitness proportionate, the probability of crossover is 0.9, the probability of mutation is 0, the probability of reproduction is 0.1, the initialization method is ramped half-and-half, the initial max depth is 6, the max depth is 17 and elitist strategy is not used. Other researches set up ex-

Table 1.1: Summary of Evolutionary Parameter Values

Parameters	Papers								
	[49, 38]	[79, 78]	[54, 53]	[124]	[93]	[82]	[112]	[14]	[15]
Population size	500	500	1024	500	1024	500	500	512	100
Generations	50	50	250	100	100	100	100	100	100
Crossover probability	0.9	0.9	0.9	0.85	0.9	0.9	0.7	0.9	0.9
Mutation probability	0.0	0.05	0.1	0.1	0.1	0.1	0.3	0.1	0.1
Initial Max depth	6	6	6	4	6	6	6	6	6
Max depth (nodes)	17	15	17	50 nodes	17	15	20	17	17
Trials per treatment	30	100	100	500	30	30	30	100	50
Elitism	Not use	1	-	1	-	-	1	10	-

perimental GP parameters often followed this parameter set. Table 1.1 summarizes the parameter sets of several recent researches. However, in most of these researches, Tournament selection is utilized instead of using Fitness proportionate for the selection method. In the dissertation, GP parameters are set with typical values that are the most common values set to GP parameters in recent researches.

1.1.8. GP benchmark problems

In GP researches, the selection of benchmark problems for evaluating the effectiveness of algorithms is also an important issue. Good, standardized benchmark problems allow direct comparisons of best results among GP methods, and between GP and non-GP techniques. Moreover, benchmark problems would help to identify the state of the art, the strengths and weaknesses of different approaches [120].

Previous researches often apply GP on regression problems. There are many benchmark regression problems that have been proposed. Table 1.2 gives information of several benchmark regression problems [120]. In the experiments of the dissertation, we selected a larger set of regression problems that are GP benchmark problems recommended in the literature [120] and regression problems taken from UCI machine learning repository [4].

1.2. Some Variants of GP

The GP community has developed many different approaches to evolve computer programs. Interestingly, these variants do not lose the basic mechanism of GP. This section briefly introduces some popular variants,

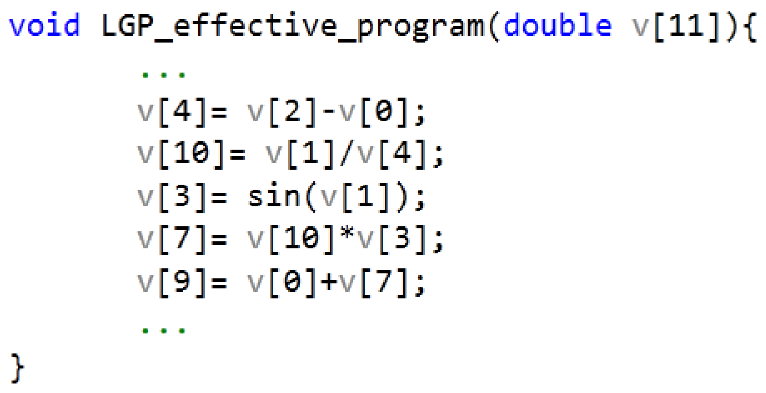
Table 1.2: GP benchmark regression problems. Variable names are, in order, x, y, z, v and w. Several benchmark problems intentionally omit variables from the function. In the training and testing sets, $U[a, b]$ is uniform random samples drawn from a to b inclusive, and $E[a, b]$ is a grid of points evenly spaced from a to b inclusive

Name	Variables	Objective function	Training set	Testing set
Koza-1	1	$x^4 + x^3 + x^2 + x$	U[-1, 1]	None
Koza-2	1	$x^5 - 2x^3 + x$	U[-1, 1]	None
Koza-3	1	$x^6 - 2x^4 + x^2$	U[-1, 1]	None
Nguyen-1	1	$x^3 + x^2 + x$	U[-1, 1]	None
Nguyen-3	1	$x^5 + x^4 + x^3 + x^2 + x$	U[-1, 1]	None
Nguyen-4	1	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U[-1, 1]	None
Nguyen-9	2	$sin(x) + sin(y^2)$	U[0, 1]	None
Nguyen-10	2	$2sin(x)cos(y)$	U[0, 1]	None
Korns-1	5	$1.57 + 24.3v$	U[-50, 50]	U[-50, 50]
Korns-2	5	$0.23 + 14.2\frac{y+v}{3w}$	U[-50, 50]	U[-50, 50]
Korns-3	5	$-5.41 + 4.9\frac{v-x+\frac{y}{w}}{3w}$	U[-50,50]	U[-50, 50]
Korns-4	5	$-2.3 + 1.3sin(z)$	U[-50, 50]	U[-50, 50]
Korns-11	5	$6.87 + 11cos(7.23x^3)$	U[-50, 50]	U[-50, 50]
Korns-12	5	$2 - 2.1cos(9.8x)sin(1.3w)$	U[-50, 50]	U[-50, 50]
Korns-13	5	$32 - 3\frac{tan(x)tan(z)}{tan(y)tan(v)}$	U[-50, 50]	U[-50, 50]
Korns-14	5	$22 - 4.2(cos(x) - tan(y))\frac{tanh(z)}{sin(v)}$	U[-50, 50]	U[-50, 50]
Korns-15	5	$12 - 6\frac{tan(x)}{e^y}(ln(z) - tan(v))$	U[-50, 50]	U[-50, 50]
Keijzer-11	2	$xy + sin((x-1)(y-1))$	U[-3, 3]	E[-3, 3]
Keijzer-13	2	$6sin(x)cos(y)$	U[-3, 3]	E[-3, 3]
Keijzer-14	2	$\frac{8}{2+x^2+y^2}$	U[-3, 3]	E[-3, 3]
Keijzer-15	2	$\frac{x^3}{5} + \frac{y^3}{2} - y - x$	U[-3, 3]	E[-3, 3]
Vladislavleva-2	1	$e^{-x}x^3cos(x)sin(x)(cos(x)sin^2(x) - 1)$	E[0.5, 10]	E[-0.5, 10.5]
Vladislavleva-4	5	$\frac{10}{5+(x-3)^2+(y-3)^2+(z-3)^2+(v-3)^2+(w-3)^2}$	U[0.05, 6.05]	U[-0.25, 6.35]
Vladislavleva-6	2	$6sin(x)cos(y)$	U[0.1, 5.9]	E[-0.05,6.05]
Vladislavleva-7	2	$(x-3)(y-3) + 2sin((x-4)(y-4))$	U[0.05,6.05]	E[-0.25,6.35]
Vladislavleva-8	2	$\frac{(x-3)^4-(y-3)^3-(y-3)}{(y-2)^4+10}$	U[0.05,6.05]	E[-0.25,6.35]

including *Linear Genetic Programming* and *Cartesian Genetic Programming*. Moreover, our proposed GP variant called *Multiple Subpopulations GP* (shortened as MS-GP) is also presented briefly in this section.

1.2.1. Linear Genetic Programming

Linear Genetic Programming (LGP) is a variant of genetic programming wherein computer programs in a population are represented as a variable-length sequence of instructions from imperative programming language or machine language [87, 97]. Instructions operate on one or two indexed variables v (also called registers) or on constants c from predefined sets. The result is assigned to a destination register. The size of the effective code varies from 0 to the number of instructions in the LGP program. Figure 1.4 demonstrates an example of LGP program.



```
void LGP_effective_program(double v[11]){
        ...
        v[4]= v[2]-v[0];
        v[10]= v[1]/v[4];
        v[3]= sin(v[1]);
        v[7]= v[10]*v[3];
        v[9]= v[0]+v[7];
        ...
}
```

Figure 1.4: An example of LGP program.

LGP allows to evolve programs in a low-level language, so these programs run directly on the computer processor. Hence, the programs can avoid the need of an interpreter and be evolved very quickly in this way. In addition, LGP can be used to solve problems with multiple outputs.

1.2.2. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a GP technique in which a program is symbolized as an indexed graph [67, 70, 87]. In CGP, the graph nodes are represented in a Cartesian coordinate system. A CGP node contains a function and pointers toward nodes representing function parameters. Each node has an output that may be used as an

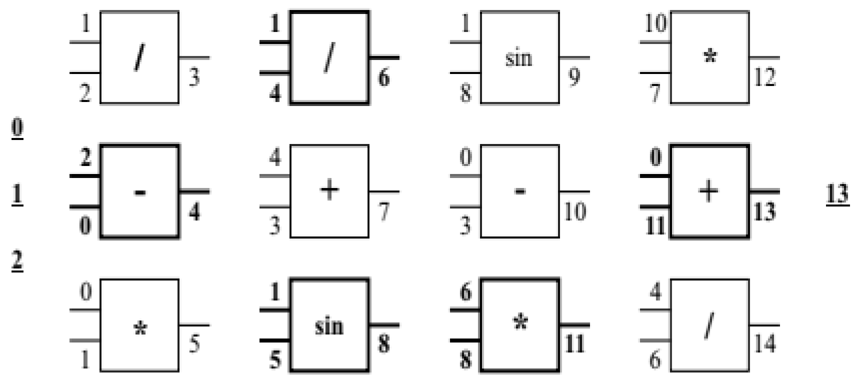


Figure 1.5: An example of CGP program.

input for another node in the graph. Nodes in the same column are not allow to be connected to each other, and any node may be either connected or disconnected. Figure 1.5 illustrates an example of CGP program with 3×4 architecture. The program has three inputs (`0,1,2`), one output (`13`) and five functions (`+, -, *, /, sin`).

Similarly to LGP, CGP is possible to have as many outputs as necessary whereas in standard GP, the evolved program has only one output.

1.2.3. Multiple Subpopulations GP

Recently, we have proposed a variant of GP that is called *Multiple Subpopulations GP* and shortened as MS-GP. While LGP and CGP change the representation of individuals, MS-GP adjusts the framework of the evolution process without losing the basic structure of GP. MS-GP divides the whole evolutionary process into two phases (Figure 1.6). In the first phase, the subdatasets are sampled from the training (full) dataset, and these subdatasets are then used to evolve the subpopulations. The individuals in the subpopulations are trained to fit a part of the training data and then combined to form the full population for the next

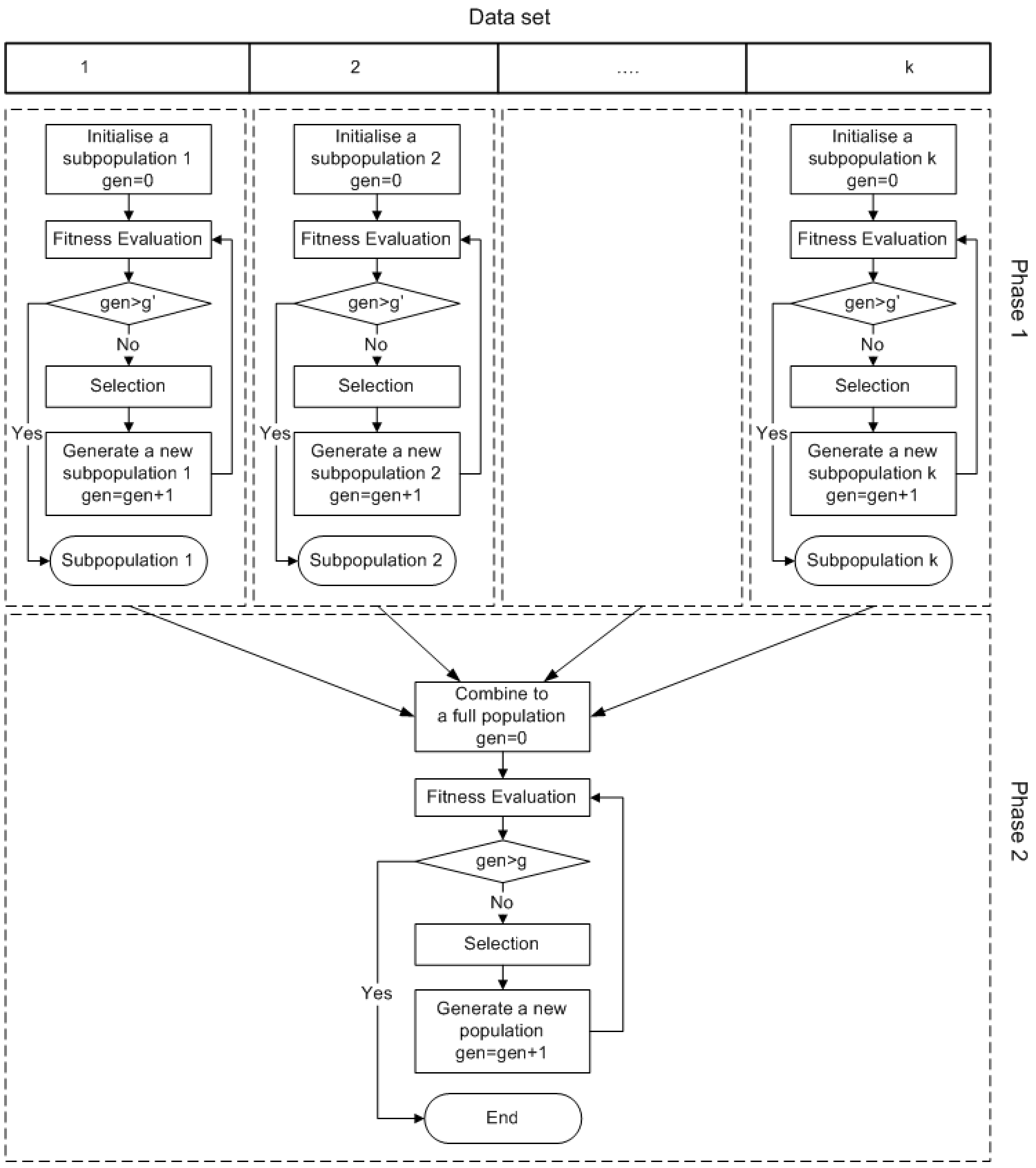

Data set
1
2
....
k
Initialise a subpopulation 1 gen=0
Fitness Evaluation
gen>g'
No
Selection
Yes
Generate a new subpopulation 1 gen=gen+1
Subpopulation 1
Initialise a subpopulation 2 gen=0
Subpopulation 2
Initialise a subpopulation k gen=0
Subpopulation k
Phase 1
Combine to a full population gen=0
gen>g
Generate a new population gen=gen+1
End
Phase 2

Figure 1.6: Structure of MS-GP.

phase. In the second phase, the combined population is evolved on the full dataset as standard GP until the last generation.

A detailed description of MS-GP is given in [C4]. We tested MS-GP on a number of regression problems and compared its performance to standard GP and some recently sampling techniques. The experimental results show that MS-GP achieved better performance compared to other tested methods. Particularly, the training error and the size of the solutions found by MS-GP are often significantly better than those found by others.

1.3. Semantics in GP

Over the last few years, semantics has been an intensively studied topic in GP. These researches applied semantic, also called phenotypic or behavioral, information for all stages of the GP evolution in a variety of ways, such as modifying or designing new genetic operators, and proposing new selection methods. This section presents the definition of program semantics in GP, and a survey of studies on semantics-based GP.

1.3.1. GP Semantics

Semantics is a broad concept used in different research fields. Generally, semantics refers to the meaning of language constructs. In ontology, semantics provides the rules for interpreting the syntax which do not provide the meaning directly but constrains the possible interpretations of what is declared [25]. In programming languages, semantics refers to the execution of the programs, i.e., what would happen each time such a program is fed into an appropriate computer [22, 36]. This means that semantics can be shown by describing the relationship between the input and output of a program, or an explanation of how the program will be executed on a certain platform.

In the context of GP, we are mostly interested in the behavior of the individuals (what they 'do'). To specify what individual behavior is, researchers have recently introduced several concepts related to semantics in GP [67, 82, 92, 93]. This subsection provides the formal concepts for program semantics in GP.

Let $p \in \mathbb{P}$ be a program from a set $\mathbb{P}$ of all possible programs. When applied to an input $in \in \mathbb{I}$, a program p produces certain output $p(in)$. In this way, a program realizes certain mapping from the set of inputs $\mathbb{I}$ into the set of outputs $\mathbb{O}$, which we denote as $p : \mathbb{I} \longrightarrow \mathbb{O}$.

Definition 1.1. *A **semantic space** of a program set $\mathbb{P}$ is a set $\mathbb{S}$ such that a semantic mapping exists for it, i.e., there exists a function $s : \mathbb{P} \rightarrow \mathbb{S}$ that maps every program $p \in \mathbb{P}$ into its semantics $s(p) \in \mathbb{S}$ and has the following property:*

$$s(p_1) = s(p_2) \Leftrightarrow \forall in \in \mathbb{I} : p_1(in) = p_2(in)$$

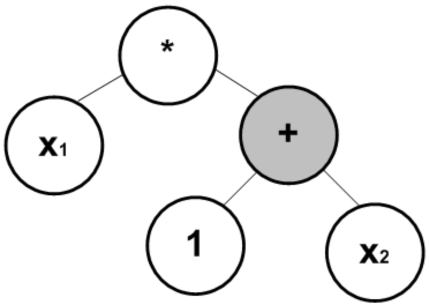
Definition 1.1 indicates that each program in $\mathbb{P}$ has thus exactly one semantics, but two different programs can have the same semantics. For example, two programs `(sin(2*x))` and `(2*sin(x)*cos(x))` have the same semantics. Moreover, programs that behave differently (i.e., produce different outputs for one or more inputs) have different semantics.

The semantic space $\mathbb{S}$ enumerates all behaviors of programs for all possible input data. That means semantics is complete in capturing the entire information on program behavior. In GP, semantics is typically contextualized within a specific programming task that is to be solved in a given program set $\mathbb{P}$. As a machine learning technique, GP evolves programs based on a finite training set of fitness cases [54, 71, 116]. Assuming that this set is the only available data that specifies the desired outcome of the sought program, naturally, an instance of the semantics of a program is the vector of outputs that the program produces for these fitness cases as Definition 1.2.

Definition 1.2. *Let $\mathbb{K} = \{k_1, k_2, ...k_n\}$ be the fitness cases of the problem. The semantics $s(p)$ of a program p is the vector of output values obtained by running p on all fitness cases.*

$$s(p) = (p(k_1), p(k_2), \dots, p(k_n)), \text{ for } i = 1, 2, \dots, n.$$

For example, the semantics of the program $x_1(1+x_2)$ in Figure 1.7 with the fitness cases as $\{(0, 0; \mathbf{0}); (0.1, 0.5; \mathbf{0.1}); (0.2, 1; \mathbf{0.4}); (0.3, 1.5; \mathbf{0.7}); (0.4, 2; \mathbf{1.2}); (0.5, 2.5; \mathbf{1.7}); (0.6, 3; \mathbf{2.4}); (0.7, 3.5; \mathbf{3.2}); (0.8, 4; \mathbf{4}); (0.9, 4.5; \mathbf{5})\}$ is the vector (0, 0.15, 0.4, 0.75, 1.2, 1.75, 2.4, 3.15, 4, 4.95).



x_1	x_2	y	p
0	0	0	**0**
0.1	0.5	0.1	**0.15**
0.2	1	0.4	**0.4**
0.3	1.5	0.7	**0.75**
0.4	2	1.2	**1.2**
0.5	2.5	1.7	**1.75**
0.6	3	2.4	**2.4**
0.7	3.5	3.2	**3.15**
0.8	4	4	**4**
0.9	4.5	5	**4.95**

Figure 1.7: Running the program p on all fitness cases

In Definition 1.2, semantics may be viewed as a point in $n-$dimensional semantic space, where n is the number of fitness cases. The semantics of a program consists of a finite sample of outputs with respect to a set of training values. Hence, this definition is not a complete description of the behavior of programs, and it is also called *sampling semantics* [78, 79]. Moreover, the definition is only valid for programs whose output is a single real-valued number, as in symbolic regression. However, this definition is widely accepted and extensively used for designing many new techniques in GP [54, 67, 73, 79, 82, 93, 110]. The studies in the dissertation use this definition.

Formally, a semantic distance between two points in a semantic space is defined as Definition 1.3.

Definition 1.3. *A* ***semantic distance*** *between two points in the semantic space $\mathbb{S}$ is any function:*

$$d : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^+$$

that is non-negative, symmetric, and fulfills the properties of the identity of indiscernibles and triangle inequality.

Interestingly, the fitness function is some kind of distance measure. Thus, semantics can be computed every time a program is evaluated, and it is essentially free to obtain. Moreover, a part of tree program is also a program, so semantics can be calculated in (almost) every node of the tree. Based on Definition 1.2, the semantic distance may be calculated by Euclidean or Manhattan distance. Let $s(p) = (p_1, p_2, ..., p_n)$ and $s(q) = (q_1, q_2, ..., q_n)$ be the semantics of the individual (program)

p and of the individual q then Euclidean distance is calculated as in Equation 1.3, and Manhattan distance is calculated as in Equation 1.4.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \tag{1.3}$$

$$d(p,q) = |p_1 - q_1| + |p_2 - q_2| + ... + |p_n - q_n| \tag{1.4}$$

Also based on Definition 1.2, the error vector of an individual is calculated by comparing the semantic vector with the target output of the problem. More precisely, the error vector of an individual is defined as:

Definition 1.4. *Let $s(p) = (s_1, s_2, ...s_n)$ be the semantics of an individual p and $y = (y_1, y_2, ...y_n)$ be the target output of the problem on n fitness cases. The error vector $e(p)$ of a program p is a vector of n elements calculated as follows.*

$$e(p) = (|s_1 - y_1|, |s_2 - y_2|, \ldots, |s_n - y_n|)$$

Overall, semantics indicates the behavior of a program (individual) and can be represented by program outputs with all possible inputs.

1.3.2. Survey of semantic methods in GP

A number of methods that incorporate semantic information into GP have been proposed in recent years. According to Leonardo Vanneschi et al. [116] these methods can be classified into two categories: indirect and direct. Indirect semantic methods achieve desired semantic criteria by sampling and rejecting individuals. Some of these methods use genetic operators to create new individuals. These individuals are then accepted and put into the new population only if the survival criteria are met [8, 7, 51, 76, 77, 78, 79, 93]. Other methods rely on semantic criteria which is applied in selection methods to select individuals as in [29, 37, 38, 55, 83].

Conversely, direct semantic methods directly act on the semantics of individuals to achieve the desired semantics. In these methods, new semantic operators are usually proposed to generate offspring. The offspring are created by a convex combination of their parents [73, 82] or by replacing subtrees in their parents with subprograms taken from a pre-defined library [52, 53, 92].

a) Indirect semantic methods:

Indirect semantic methods operate on the syntax of the individuals and rely on survival criteria to indirectly promote a semantic behavior. These survival criteria often focus on one of three aspects of the evolution: semantic diversity, locality and geometry of the landscape [116].

McPhee et al. [67] are probably the first researchers to define the semantics of an individual as its outputs with respect to a set of inputs. The semantics of a boolean expression tree is extracted into truth tables by enumerating all possible inputs. The truth tables are then used to analyze the behavioral changes of the two components (the subtrees and the context) on each tree during the evolutionary process. Special attention was paid to fixed-semantic contexts: contexts such that the semantics of the entire tree does not change when subtrees are replaced by other subtrees. For example, the context `AND(false,subtr)` returns a stable value of `false` with every replacement subtree, `subtr`. It emerges that there may be many such fixed semantic contexts in a boolean domain problem, especially when the size of the tree increases during evolution. At that point, it is difficult to change the semantics of trees with crossover and mutation. Therefore, the distribution of context semantics is key to the success (or failure) of runs.

In the same boolean problems, Beadle and Johnson [6] proposed a new crossover operator called Semantically-Driven Crossover (SDC). SDC uses an efficient technique called Reduced Ordered Binary Decision Diagrams (ROBDDs) to measure behavioral difference between two individuals. Any two individuals that reduce to the same ROBDD are semantically equivalent. SDC checks the semantic equivalence of the offspring with their parents and discards the offspring if equivalent to their parents. The operator increases semantic diversity in the evolutionary population and thus leads to improved performance of GP. This technique was also applied in [8] to drive behavior of individuals at mutation and in [7] to guide the initialization population in GP, with a positive effect on performance.

Next, Nguyen et al. [77] investigated the role of semantic diversity and locality of crossover operators for boolean problems. Two crossover operators were proposed: Guaranteed Change Semantic Crossover (GCSC) to promote semantic diversity and Locality Controlled Semantic Crossover (LCSC) to improve semantic locality. GCSC guarantees a change in the semantics of children in the new population by aborting children with the same semantics as their parents. LCSC is an extension of GCSC that is aborted not only if the offspring has the same semantics, but also if it is too different from their parents.

For real-valued problems, it is not easy to exactly calculate the semantics or equivalent semantics of two tree representations by reducing them to a common structure, since this is a problem of NP-hard class [33, 116]. Instead, semantics is evaluated on a set of sampling fitness cases in the domain of the problem. Nguyen et al. [76] proposed a crossover operator called Semantics Aware Crossover (SAC). The approach of SAC can apply to both boolean and real-valued domain problems. This crossover operator also aims to encourage semantic diversity. SAC prevents an exchange between two subtrees with the semantic distance between them smaller than a given threshold when doing crossover. SAC was then extended to Semantic Similarity based Crossover (SSC) [79] and Most Semantically Similar Crossover (MSSC) [78]. SSC performs crossover when two subtrees are semantic similarity, which means the semantic distance between these subtrees within the interval $[\alpha, \beta]$. Additionally, if a pair of semantically similar trees could not be found after the maximum number of trials, SSD uses standard crossover. In MSSC, it selects randomly n pair of subtrees from two parents and calculates the semantic distance between each pair. Pairs with semantic equivalence are excluded. In the remaining pairs, the pair with the smallest semantic distance is selected and used for crossover. The results illustrated that this approach increases semantic diversity in the evolutionary population, and thus it leads to the improvement in the GP performance.

Based on the geometry of landscape, Krawiec et al. [51, 93] proposed a new crossover operator for GP, namely Krawiec and Lichocki Geometric Crossover (KLX). KLX uses standard crossover to generate a number of candidate offspring. The offspring with the smallest total distance from their parents calculating by Equation 1.5 are added to the next generation. This operator is characterized by high probability of producing offspring semantically equal to one of their parents [92].

$$\underbrace{d(s(p), s(p_1)) + d(s(p), s(p_2))}_{\text{distance sum}} + \underbrace{|d(s(p), s(p_1)) - d(s(p), s(p_2))|}_{\text{penalty}} \qquad (1.5)$$

where p_1 and p_2 are parents, p is an offspring and $d(.)$ is a function distance.

Semantic information is also incorporated into the selection mechanism of GP. These methods are based on semantic criteria to select parents, and they are surveyed in detail in Subsection 1.3.3.

Overall, the indirect methods often lead to the promotion of semantic diversity and local semantics of a GP population, thereby improving GP performance. However, these methods involving the process of rejecting a number of individuals during the GP evolution are slow in the GP evolution [51, 76, 78, 79].

b) Direct semantic methods:

Direct semantic methods act directly on the semantics of individuals to effectively achieve the desired semantic goal [116]. Moraglio et al.[73] proposed a new approach based on semantics to design geometric semantic operators, called Geometric Semantic Genetic Programming(GSGP). The geometric semantic operators generate offspring by a convex combination of its parents. Two operators were introduced, including SGX and SGM. SGX is a crossover operator that generates a offspring t from two parents p_1 and p_2 by Equation 1.6 for a boolean domain problem and Equation 1.7 for a real domain problem.

$$t = (p_1 \wedge t_r) \vee (\bar{t_r} \wedge p_2) \tag{1.6}$$

where t_r is a randomly generated Boolean function.

$$t = (p_1.t_r) + ((1 - t_r).p_2) \tag{1.7}$$

where t_r is a constant or a function randomly generated with codomain in the interval $[0, 1]$.

Experimental results showed that these operators achieved a good performance with the experimental problems. However, the geometric semantic operators can be very time- and memory-consuming since they keep all parents in memory, accumulate their complexity (number of nodes) via the convex combination.

In order to address the limitations of GSGP, many studies have been recently proposed [11, 35, 72, 82, 84, 85, 86, 114, 117]. Nguyen et al. [82] proposed a new semantic crossover called SSGX which is very similar to the SGX operator except that is installed on the subtree level. SSGX finds a subtree with the most semantic similarity with its parent, and the subtree is then used instead of its parent in the convex combination to generate offspring. Martins et al. [64] solved the exponential growth of GSGP by a proposed method called *GSGP with Reduced trees* (GSGP-Red). GSGP-Red expands the functions representing the individuals into linear combinations, and then aggregates the repeated structures. Experimental results indicated that these operators reduced code growth phenomenon and increased GP performance. Next, Oliveira et al. [86]

proposed the Geometric Dispersion operator (GD) with the goal of dispersing semantics of individuals in the population around the target output point. GD operator is utilized before using other operators in GSGP. Experimental results indicated these operators help improve the efficiency of GSGP.

Krawiec et al. [52, 54] proposed a new geometric crossover called Locally Geometric Semantic Crossover (LGX). The idea of LGX is to introduce a semantic combinatorial geometry. LGX randomly takes two subtrees in the common shape from their parents, and calculates the desired semantics of offspring as the midpoint of the semantics of these subtrees. LGX then searches for a subprogram in a pre-defined library such that its semantics is the closest to this desired semantics, and the selected subtrees are replaced by this subprogram to generate offspring.

Continually, Krawiec et al. [53] proposed a semantic backpropagation algorithm that calculates the desired semantics of an intermediate node. The algorithm assigns the desired semantics of a program to the root node, and then propagates this value reversely down the program tree to determine the desired semantics of intermediate node. Based on this algorithm, two new genetic operators, AGX and RDO, were proposed [92]. The main idea of AGX is to replace subtrees on their parents with subprograms taken from a pre-defined library such that the offspring are semantically intermediate to their parents. AGX calculates the midpoint of the semantics of their parents and randomly selects two crossover points in the parents. The semantic backpropagation algorithm is then used to determine desired semantics at the crossover points. After obtaining the desired semantics, AGX finds in a pre-defined library a subprogram which is the semantically closest to the desired semantics to substitute the subtrees. RDO is the form of a mutation operator. It uses the semantic backpropagation algorithm as AGX, but the desired semantics is the target semantics of the problem.

Recently, Pawlak and Krawiec [91], in turn, analysed a wide range of mutation and crossover operators under different metrics and proposed some competent operators, including Competent Mutation (CM) and Competent Crossover (CX). CM operator is similar to RDO [92] except for the condition of finding an alternative subtree in the pre-defined library in addition to ensuring the closest to the desired semantics must also change the semantics of the parent. CX is an extension of AGX with the additional condition that finds the replacement subtree to create a semantic change of parents. Experimental results indicated that these operators achieved the good performance in comparison to standard crossover except for high computational costs.

More recently, Chen et al. [14, 15, 16] incorporated the angle-awareness into GSGP and proposed some new methods such as Angle-aware Geometric Semantic Crossover (AGSX), Perpendicular Crossover (PC) and Random Segment Mutation (RSM). AGSX [14] first selects randomly the first parent and then finds the second parent from the candidate set so that it maximizes the angle between the relative semantics of the two parents to the target output. After that, two parents are used to create offspring as LGX [54]. Combining the angle-awareness and the semantic backpropagation algorithm, PC and RSM [15, 16] first determine the desired semantic of offspring. The desired semantics of the offspring in PC is the perpendicular projection of the target output to the semantics of two parents, and the desired semantics of the offspring in RSM is the midpoint between the semantics of the parent and the target output. After that, the remaining process of these operators is similar to that in RDO. Experimental results show that these operators improve the performance of GP. Nevertheless, the methods using the semantic backpropagation algorithm often calculate the desired semantics and then compare it with that of the subprograms in a pre-defined library to find a semantic approximation tree. This leads to the slow GP evolution [14, 53, 92, 110].

Overall, this subsection has reviewed semantic methods and showed that these methods have resulted in the increase in semantic diversity, local semantics of the population and GP performance.

1.3.3. Semantics in selection and control of code bloat

The works using semantic information in the selection process and bloat control in GP are few as reported in Figure 2. Galvan-Lopez et al.[29] proposed a technique called Semantic in Selection (SiS) for selecting semantically different parents. SiS selects two parents. The first parent is selected normally by using standard tournament selection. The second parent is then selected by applying standard tournament selection on only individuals whose semantics is different from that of the first parent. In other words, only individuals with the semantic distance to the first parent greater than a given threshold can participate in the tournament to select the second parent.

Next, Pawlak and Krawiec [91] proposed a new selection method for GSGP called Competent Tournament Selection (CTS) to select two parents. As SiS, CTS chooses the first parent also using standard tournament selection, and the second parent is selected by minimizing the custom distance of individual candidates.

For controlling bloat in GP, Fracasso and Zuben [28] recently proposed two mutation operators, Multi-Objective Randomized Similarity Mutation (MORSM) and Multi-Objective Desired Operator(MODO), to avoid the bloating effect. In MORSM operator, a random subtree of a parent is selected and replaced by a random subprogram chosen in a Restricted Candidate List (RCL). RCL consists of subprograms that are taken from a pre-defined library and are non-dominated for two objectives. The first objective is minimum the semantic distance between the selected subtree and the subprograms, and the second objective is minimum the size of the subprograms.

MODO is an extension of RDO operator [92], aiming at controlling code bloat. In MODO, after randomly selecting a subtree in the parent, the desired semantics of this subtree is calculated by using the semantic backpropagation algorithm. Then, the restricted list with non-dominated subprograms, RCL is built. RCL aims at minimizing the distance between the desired semantics of the selected subtree and the semantics of the subprograms, and minimizing the size of the subprograms. Finally, a subprogram of RCL is chosen randomly and utilized to replace the selected subtree.

1.4. Semantic Backpropagation

The semantic backpropagation algorithm was proposed by Krawiec and Pawlak [53, 93] to determine the desired semantics for an intermediate node of an individual. The algorithm starts by assigning the target of the problem (the output of the training set) to the semantics of the root node and then propagates the semantic target backwards through the program tree. At each node, the algorithm calculates the desired semantics of the node so that when we replace the subtree at this node by a new tree that has semantics equally to the desired semantics, the semantics of entire individual will match the target semantics.

Figure 1.8 illustrates the process of using the semantic backpropagation algorithm to calculate the desired semantics for the blue node N. In this figure, {(1,5);(2,8);(3,16)} is the fitness cases, and the semantic vector of each node is calculated and displayed in the solid box. The

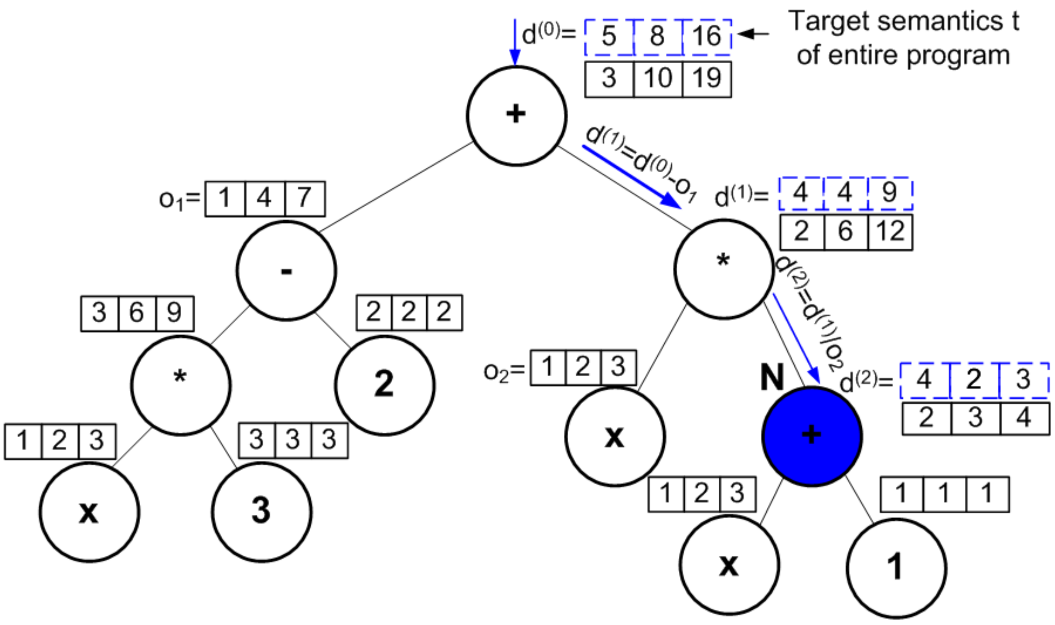


Figure 1.8: An example of calculating the desired semantics of the selected node N.

target semantic vector is (5,8,16) and the desired semantics of the blue node N and its parents are calculated and presented in the dashed box.

The semantic backpropagation technique is then used for designing several genetic operators in GP [53, 93]. Among these, Random Desired Operator (RDO) is the most effective operator [82, 93]. A parent is selected by a selection scheme, and a random subtree *subTree* is chosen. The semantic backpropagation algorithm is used to identify the desired semantics of *subTree*. After that, a procedure is called to search in a pre-defined library of trees for a *newTree* that is the most semantically closest to the desired semantics. Finally, *newTree* is replaced for *subTree* to create a new individual. This operator performs very well on both real-valued and Boolean problems as reported in [82, 93].

1.5. Statistical Hypothesis Test

In Chapter 2, we propose the use of a statistic hypothesis test to analyse the error vectors of individuals competing in a tournament. Practically, a Wilcoxon signed-rank test is employed the experiments. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples to assess whether their population mean ranks differ [43]. This test is often used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed.

Let n be the sample size of the test and $x_{1,i}$ and $x_{2,i}$ denote the i^{th} pair sample. Let H_0: be the hypothesis that difference between the pairs follows a symmetric distribution around zero and H_1: be the hypothesis that difference between the pairs does not follow a symmetric distribution around zero. The test is performed as follows:

1. For $i = 1, ..., n$, calculate $|x_{2,i} - x_{1,i}|$, and $sgn(x_{2,i} - x_{1,i})$, where sgn is the sign function:

$$sgn(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases} \tag{1.8}$$

2. Exclude pairs with $|x_{2,i} - x_{1,i}| = 0$. Let n_r be the reduced sample size.

3. Order the remaining n_r pairs from smallest absolute difference to largest absolute difference, $|x_{2,i} - x_{1,i}|$.

4. Rank the pairs, starting with the smallest as 1. Ties receive a rank equal to the average of the ranks they span. Let R_i denote the rank.

5. Calculate the test statistic W, the sum of the signed ranks:

$$W = \sum_{i=1}^{n_r} [sgn(x_{2,i} - x_{1,i}) \cdot R_i] \tag{1.9}$$

6. The value of W is compared to a threshold to decide if the null hypothesis H_0 is rejected.

In practice, the *p_value* is often calculated from the test. The *p_value* is defined as the probability of obtaining a result equal to or more extreme than what was actually observed, when the null hypothesis is true [100]. If *p_value* is smaller than a threshold (called the critical value), the null hypothesis is rejected and two data set are significantly different. Otherwise, we can not reject the null hypothesis. In statistics, two popular critical values of *p_value* (0.01 and 0.05) are often used. These values correspond to the confident level of 99% and 95% to reject the null hypothesis, respectively.

1.6. Conclusion

This chapter presents the backgrounds of GP. Firstly, a more detailed introduction has been given. The discussion of GP includes algorithm, initialisation, fitness evaluation, selection, operators and parameters. Secondly, the dissertation presents some variants of GP and a new proposed variant for GP. Next, some foundation concepts of the dissertation including the semantics of a program, semantic distance and error vectors of the program as well as the survey of semantic methods in GP are also presented. Then, the semantic backpropagation algorithm is introduced. Finally, the chapter presents a statistical hypothesis test.

The survey has shown the effectiveness and limitations of semantic integration methods in GP. In order to continue promoting the effectiveness and to soften the drawbacks when incorporating semantic information into GP, the dissertation presents several proposed techniques that incorporate semantics into GP, including both indirect and direct methods. Chapter 2 introduces the new indirect methods that use a statistical test on semantic information from the individual's error vector into the GP tournament selection. Chapter 3 presents the direct methods that integrate semantic information to reduce code bloat in GP.

Chapter 2
TOURNAMENT SELECTION USING SEMANTICS

The chapter introduces the use of a statistical test into GP tournament selection that utilizes information from the individual's error vector, and three variants of the selection strategy are proposed. These methods are tested on a larger number of regression problems and their noisy variants. The experimental results demonstrate the benefit of the proposed methods in reducing GP code growth and improving the generalisation behaviour of GP solutions when compared to standard tournament selection, a similar selection technique and a state of the art bloat control approach. The results in this chapter have been published in [C1, C3].

2.1. Introduction

There are several factors that can effect the performance of GP for a given problem. These factors include the size of the population, the fitness evaluation of individuals, the selection mechanisms for reproduction and the genetic operators for modifying individuals. Amongst these, selection plays a critical role in GP performance [10]. To date, there have been many selection schemes proposed [48] and the most widely used selection in GP is tournament selection [26].

Tournament selection compares the fitness value of sampled individuals. The individual with the best fitness is then selected as the winner. This implementation is simple, and its effectiveness has been widely evidenced [26]. However, the standard approach only uses the fitness value while ignoring finer-grained semantics which can be collected during GP program execution. Consequently, some information that is potentially useful for GP search may be lost. Recent research has shown that significant benefit could be gained by using semantic information of GP individuals (e.g., [44, 45, 74, 79, 93]). The genetic search operators of crossover and mutation can be modified to improve the semantic locality of search [23, 78, 90]. In addition, the preservation of semantic diversity is a desirable feature of an evolving GP population to avoid local optima [12, 27], thus, it is also attractive to examine whether using the error vector of individuals on the fitness cases during selection can improve GP performance.

This chapter presents to use statistical testing techniques with individual error vectors to improve the effectiveness and efficiency of GP. The main contributions of this chapter are:

- The chapter introduces the use of statistical analysis of GP error vectors to create novel forms of tournament selection. Based on a Wilcoxon signed rank test, three variants of tournament selection are proposed to exploit semantic diversity and to explore the potential of the approach to control program bloat.
- The performance of the selection strategies are examined on a large set of regression problems employing the original problems and noisy

variants. We observe that the new selection techniques help to reduce the code growth and improve the generalization ability of the evolved solutions when compared to standard tournament selection and a state of the art method for controlling code bloat in GP.

- The simplicity of the design of the proposed selection strategies allows for further improvements. In this chapter, the addition of a state of the art crossover operator is observed to further enhance performance.

The rest of this chapter is organised as follows. The next section reviews the related work on improving tournament selection in GP. Three proposed tournament selection strategies are presented in Section 2.3. Section 2.4 presents the experimental settings adopted in the chapter. Section 2.5 analyses, compares and improves the performance of the proposed selection strategies. Finally, Section 2.6 concludes the chapter.

2.2. Tournament Selection Strategies

Selection is a key factor that affects the performance of Evolutionary Algorithms (EAs) [1, 24]. Commonly used parent selection strategies in EAs include fitness proportionate selection, rank selection, and tournament selection [10]. The most popular selection method in GP is tournament selection [123]. In standard 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 [10].

The advantage of tournament selection is that it allows the adjustment of the selection pressure by tuning the tournament size. A small value of the 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 leading to saving a large amount of the processing time [125].

Since tournament selection is the most popular selection method in GP, there have been many studies to analyse its behaviour and improve its effectiveness. As the selection process of standard tournament selection consists of sampling and selecting, the majority of the early studies have focused on sampling and selecting [26].

2.2.1. Sampling strategies

Gathercole et al. [32] analysed the selection frequency of each individual and the likelihood of not-selected and not-sampled individuals in tournament selection with different values of the tournament size. Sokolov and Whitley proposed unbiased tournament selection [108] where all individuals have a fair chance to participate in a tournament.

Later, Xie and his colleagues conducted a series of studies to investigate tournament selection in GP. In [122], Xie indicated that standard tournament selection can lead to the result in which the individuals with bad fitness could be selected multiple times while the individuals with good fitness not selected any time. Thus, he proposed the fully covered tournament selection method [122] which excludes the sampled individuals in the next tournament to ensure that each individual has an equal chance to participate into tournaments. Next, Xie et al. [123, 126] analysed the performance of no-replacement tournament selection in GP. In the no-replacement strategy, no individual can be sampled multiple times into the same tournament. Another problem in tournament selection is that some individuals are not sampled at all when using small values of the tournament size. However, Xie at al. [127] showed that the not-sampled issue does not seriously affect the selection performance in standard tournament selection.

Overall, previous research has shown that sampling strategies have a minor impact on GP performance. Consequently, researchers have paid more attention to the second step in tournament selection: selection.

2.2.2. Selecting strategies

Goldberg and Deb [34] introduced binary tournament selection in which two individuals are selected at random, and the individual with better fitness could be selected with probability p, $0.5 < p \leq 1.0$. Back [5] ranked the individuals in the population in which the best individual was ranked 1^{st}. After that, the selection probability of an individual of rank j is calculated by:

$$n^{-k}((n-j+1)^k - (n-j)^k) \tag{2.1}$$

where k is size of tournament and n is size of population. Conversely, Blickle and Thiele [10] defined the worst individuals to be ranked 1^{st} and introduced the cumulative fitness distribution, $s(f_j)$, which denotes the number of individuals with fitness value f_j or worse. Finally, the selection probability of individuals with rank j is calculated as:

$$(\frac{s(f_j)}{n})^k - (\frac{s(f_{j-1})}{n})^k. \tag{2.2}$$

Next, Julstrom and Robinson proposed weighted k-tournaments method [42] in which a parameter w between 0 and 1 is chosen, and the probability that the i^{th} worst contestant is selected is proportional to w^{k-i}. More precisely, the selection probability of individuals with rank j is calculated by the formula:

$$\frac{k(1-w)}{n^k(1-w^k)}((j-1)+w(n-j))^{k-1} \tag{2.3}$$

Later, Hingee and Hutter [39] introduced the polynomial rank scheme of degree d for calculating the probability of an individual with rank j as follows:

$$P(I=j) = \sum_{t=1}^{d+1} a_t j^{t-1} \tag{2.4}$$

They also showed that every probabilistic tournament is equivalent to a unique polynomial rank scheme.

Recently, some researchers have focused on adapting the selection pressure. Xie and Zhang [124] proposed a method for automatically tuning the selection pressure during evolution based on the fitness rank distribution of the population. In each generation, they cluster the population into s clusters. Next, they sample k clusters from s clusters with replacement and selected the winner among k sampled clusters. Finally, a random individual is returned from the winning cluster. Their experimental results showed that the proposed approach is able to tune parent selection pressure automatically and dynamically along the evolution.

In an approach to control semantic locality and preserve semantic diversity during selection in grammatical evolution, Forstenlechner et al. [27] introduced semantic clustering selection. First, the individuals in a GP population are clustered based on the similarity of their error vectors. Then, parents are drawn from within the same cluster to improve semantic locality. Moreover, semantic diversity is managed through the preservation of the existence of multiple clusters.

Another method proposed by Trujillo et al. [112] that allows to eliminate the concept of tournament size is neatGP. In fact, neatGP was a recent method with the main objective is to reduce code bloat in GP. This method was inspired by neuro evolution of augmenting topologies algorithm in neural networks [113] and the flat operator equalization bloat control method in GP [21]. The detailed description of neatGP could be found in [112].

More recently, Lexicase Selection was proposed for solving uncompromising problems [38, 68]. The idea is to evaluate the goodness of an individual based on part of fitness cases instead of all fitness cases. Each time a parent must be selected, Lexicase Selection first shuffles the list of fitness cases into a random order. Next it removes any individual that did not achieve the best error value on the first fitness case. If more than one individual remains in the population, the first fitness case is removed and this process is repeated with the next fitness case. This technique was then extended to the real-valued regression problem by La Cava et al. [56] and was proved to maintain better diversity compared to standard tournament selection [37, 55, 83].

In this chapter, the dissertation introduces a new proposed method for selecting the winner in tournament selection that is based on the statistical analysis of the semantics of GP programs. Specifically, we focus our attention on the error vectors produced in symbolic regression problems. The most similar approach in the literature was Semantic in Selection (SiS) technique proposed by Galvan-Lopez et al [29] which calculates the semantic similarity of parents and selects parents which are semantically dissimilar (i.e., have large differences between their semantic vectors). Rather than computing differences in semantic vectors, we perform a statistical analysis based on error vectors to ascertain semantic diversity of the individuals competing in a tournament. A detailed description of our method will be presented in the next section.

2.3. Tournament Selection based on Semantics

In this section, we present the usage of statistical analysis of error vectors in the form of a statistical hypothesis test during tournament selection. The objective is to select breeding parents based on the statistical test instead of on their fitness values. It is difficult to assess whether the values of error vectors follow a normal distribution; thus, a non-parametric statistical hypothesis test is appropriate, and Wilcoxon Signed Rank Test is selected in our experiments. Three new selection strategies are proposed, including *Statistics Tournament Selection with Random* (TS-R), *Statistics Tournament Selection with Size* (TS-S) and *Statistics Tournament Selection with Probability* (TS-P).

2.3.1. Statistics Tournament Selection with Random

The first proposed method is called *Statistics Tournament Selection with Random* and shortened as TS-R. The main objective of TS-R is to promote the semantic diversity of GP population compared to standard tournament selection. The process of TS-R is similar to standard tournament selection. However, instead of using the fitness value for comparing between individuals in the tournament, a statistical test is applied to the error vector of these individuals. For a pair of individuals, if the test shows that the individuals are different, then the individual with better fitness value is considered as the winner. Conversely, if the test confirms that two individuals are not different, a random individual is selected from the pair.

Algorithm 3: Statistics Tournament Selection with Random

```
Input: Tournament size: $TourSize$, Critical value: $alpha$.
Output: The winner individual.
$A \longleftarrow RandomIndividual()$;
for $i \leftarrow 1$ to $TourSize$ do
    $B \longleftarrow RandomIndividual()$;
    $sample1 \longleftarrow Error(A)$;
    $sample2 \longleftarrow Error(B)$;
    $p\_value \longleftarrow Testing(sample1, sample2)$;
    if $p\_value < alpha$ then
        $A \longleftarrow GetBetterFitness(A, B)$;
    else
        $A \longleftarrow GetRandom(A, B)$;
    end
end
The winner individual $\longleftarrow$A ;
```

After that, the winner is tested against other individuals in the tournament size. This process is repeated for all individuals in the tournament size. The detailed description of TS-R is presented in Algorithm 3.

In Algorithm 3, function *RandomIndividual*() returns a random individual from the GP population. Function $Error(A)$ assigns the error vector of individual A to $sample1$ [1], and function $Testing(sample1, sample2)$ performs the Wilcoxon signed rank test. Two last functions, *GetBetter-Fitness*(A, B) and *GetRandom*(A, B) aims to find the better fitness individual among A and B or return a random individual between two, respectively. Finally, *alpha* is the critical value used to decide if the null hypothesis is rejected or accepted. If the output of the test (*p_value*) is smaller than *alpha*, then the null hypothesis is rejected. This means that two individuals are significantly different and the better fitness individual is selected as the winner. If the test can not reject the null hypothesis, then a random individual is selected from the pair.

2.3.2. Statistics Tournament Selection with Size

The second proposed tournament selection is called *Statistics Tournament Selection with Size* and shortened as TS-S. TS-S is similar to TS-R in the objective of promoting diversity. Moreover, TS-S also aims at reducing the code growth in GP population. In TS-S, if the statistical test can not reject the null hypothesis, then the individual with smaller size is selected from the pair. In other words, if two individuals involved in the test are not statistically different, then the smaller individual will

[1] In order to reduce the computational time of this method, the error vectors of all individuals are calculated at the beginning of each generation. These error vectors are then stored for using in every statistical test in the same generation.

be the winner. The detailed description of TS-S is presented in Algorithm 4. In this algorithm, function $GetSmallerSize(A, B)$ returns the individual with smaller size among A and B. If the size of A and B are tie, then the first individual will be returned.

Algorithm 4: Statistics Tournament Selection with Size

```
Input: Tournament size: $TourSize$, Critical value: $alpha$.
Output: The winner individual.
$A \longleftarrow RandomIndividual()$;
for $i \leftarrow 1$ to $TourSize$ do
    $B \longleftarrow RandomIndividual()$;
    $sample1 \longleftarrow Error(A)$;
    $sample2 \longleftarrow Error(B)$;
    $p\_value \longleftarrow Testing(sample1, sample2)$;
    if $p\_value < alpha$ then
        $A \longleftarrow GetBetterFitness(A, B)$;
    else
        $A \longleftarrow GetSmallerSize(A, B)$;
    end
end
The winner individual $\longleftarrow$A ;
```

2.3.3. Statistics Tournament Selection with Probability

The third tournament selection method is called *Statistics Tournament Selection with Probability* and shorted as TS-P. This technique is different from TS-R and TS-S in which it does not rely on the critical value to decide the winner. Instead, TS-P uses the *p_value* as the probability to select the winner. In other words, the better fitness individual is selected with the probability of $1-$ *p_value* while the worse fitness individual has the probability of *p_value* to be selected. The detailed description of TS-P is presented in Algorithm 5.

Algorithm 5: Statistics Tournament Selection with Probability

```
Input: Tournament size: $TourSize$.
Output: The winner individual.
$A \longleftarrow RandomIndividual()$;
for $i \leftarrow 1$ to $TourSize$ do
    $B \longleftarrow RandomIndividual()$;
    $sample1 \longleftarrow Error(A)$;
    $sample2 \longleftarrow Error(B)$;
    $p\_value \longleftarrow Testing(sample1, sample2)$;
    $A \longleftarrow GetBetterWithProbability(A, B, p\_value)$;
end
The winner individual $\longleftarrow$A ;
```

In this algorithm, after conducting the statistical test between A and B, function $GetBetterWithProbability(A, B, p\_value)$ return the better fitness individual between A and B with the probability of $1-$ p_value and return the worse fitness individual with the probability of p_value.

Overall, in the Statistics Tournament Selection methods, by using a statistical test, an individual is only considered as a winner if its fitness value is significantly better than other individuals, otherwise, the winner will be selected with the other criteria. As a result, the semantic diversity of the GP population is promoted. It is noted that, in order to select the winner in the three proposed tournament selection techniques, a series of the Wilcoxon signed rank test is applied on the error vectors of sampled individuals. Potentially, there are two limitations of this approach. The first limitation is the overuse of statistical tests may result in the significant difference being detected by chance as indicated by Cumming [17]. This, subsequently, may affect the performance of statistics tournament selection. However, in Subsection 2.5.1 we show that, on average, approximate 50% to 70% of the Wilcoxon test in TS-R and TS-S is significant. This large number may help to alleviate the impact of the test that is affected by chance.

The second limitation is the overhead of computational time resulting from executing a series of the statistical test. In terms of algorithm complexity, the time complexity of statistics tournament selection is $T(n)$ times the time complexity of standard tournament selection, where $T(n)$ is the time complexity of the statistical hypothesis test. In the algorithm of Wilcoxon signed rank test, $T(n)$ depends on a sorting algorithm used in step 3. $T(n)$ equals $O(n.log(n))$, where n is the number of fitness cases, when using the best sorting algorithm. The time complexity of a single tournament in standard tournament selection is $O(k)$, where k is tournament size. Thus, the time complexity of statistics tournament selection is $O(k.n.log(n))$.

Besides, in the experiment, we compare the computational time of executing the selection step in statistics tournament selection and in standard tournament selection and find that the execution time of the selection step in statistics tournament selection is greater than that of standard tournament selection. However, statistics tournament selection often helps to remarkably reduce code growth of GP population. Subsequently, the GP system that implements statistics tournament selection often runs faster than the system that uses standard tournament selection.

2.4. Experimental Settings

This section presents the problems that will be used for testing the statistics tournament selection methods that are proposed the previous section. The parameter settings for the GP systems in our experiments are also given.

2.4.1. Symbolic Regression Problems

In order to evaluate the proposed tournament selection techniques, we tested them on a large number of problems including twenty-six regression problems. Among them, fifteen problems are GP benchmark problems recommended in the literature [120] and an additional eleven problems were taken from UCI machine learning repository [4].

For each problem, we also created a noisy version from the original (noiseless) form that results in twenty-six noisy datasets. In total, fifty-two datasets were used for the experiments in this chapter. The detailed description of the tested problems including the abbreviation, their name, number of features, number of training and testing samples are presented in Table 2.1.

2.4.2. Parameter Settings

The GP parameters used for our experiments are shown in Table 2.2. They are typical values for the experiments based on GP [49]. The terminal set for each problem includes n variables corresponding to the number of features of that problem. The raw fitness is the mean of absolute error on all fitness cases. Therefore, smaller values are better. In all

Table 2.1: Problems for testing statistics tournament selection techniques

Abbreviation	Name	Features	Training	Testing
A. **Benchmarking Problems**				
F1	korns-11	5	20	20
F2	korns-12	5	20	20
F3	korns-14	5	20	20
F4	vladislavleva-2	1	100	221
F5	vladislavleva-4	5	500	500
F6	vladislavleva-6	2	30	93636
F7	vladislavleva-8	2	50	1089
F8	korns-1	5	1000	1000
F9	korns-2	5	1000	1000
F10	korns-3	5	1000	1000
F11	korns-4	5	1000	1000
F12	korns-11	5	1000	1000
F13	korns-12	5	1000	1000
F14	korns-14	5	1000	1000
F15	korns-15	5	1000	1000
B. **UCI Problems**				
F16	airfoil_self_noise	5	800	703
F17	casp	9	100	100
F18	ccpp	4	1000	1000
F19	wpbc	31	100	98
F20	3D_spatial_network	3	750	750
F21	protein_Tertiary_Structure	9	1000	1000
F22	yacht_hydrodynamics	6	160	148
F23	slump_test_Compressive	7	50	53
F24	slump_test_FLOW	7	50	53
F25	slump_test_SLUMP	7	50	53
F26	Appliances_energy_prediction	26	5000	9235

experiment, three popular values of tournament size (referred to as tour-size hereafter) including 3, 5 and 7 were tested[2]. The elitism technique was also used in which the best individual in the current generation is always copied to the next generation.

Table 2.2: Evolutionary Parameter Values.

Parameters	Value
Population size	500
Generations	100
Tournament size	3, 5, 7
Crossover, mutation probability	0.9; 0.1
Function set	$+, -, *, /, sin, cos$
Terminal set	$X_1, X_2, ..., X_n$
Initial Max depth	6
Max depth	17
Max depth of mutation tree	15
Raw fitness	mean absolute error on all fitness cases
Trials per treatment	100 independent runs for each value
Elitism	Copy the best individual to the next generation.

A popular value of the critical value in Wilcoxon test, $alpha = 0.05$, was used in TS-R and TS-S to decide if the null hypothesis is rejected. For each problem and each parameter setting, 100 runs were performed.

To compare between different methods in terms of statistics, we followed the suggestion by Derrac [19] to employ the Friedman's test on the results in all tables in the following sections. If the Friedman test shows that at least the result of one technique is significantly different from the others with confident level of 95%, we conducted a post-hoc

[2]Since the space limitation, we only show in this chapter the results with tour-size=3 and tour-size=7. The results with tour-size=5 are presented in the appendix of the dissertation and at https://github.com/chuthihuong/GP.

analysis with the Bonferroni–Dunn correction of the *p_value* for each comparison [19]. In the tables, if the result of a method is significantly better than GP with standard tournament selection (shorthanded as GP hereafter), this result is marked + at the end. Conversely, if it is significantly worse compared to GP, this result is marked $-$ at the end. In addition, if it is the best (lowest) value, it is printed underline, and if the result of a method is better than GP, it is printed bold face.

The source code of all tested methods in this chapter are available for download [3]. All techniques were implemented in Java with the exception of neatGP for which we used the implementation in Python [4]. Moreover, the same computing platform (Operating system: Windows 7 Ultimate (64bit), RAM 16.0GB, Intel ® Core™i7-4790 CPU@3.60GHz) was used in every experiment in this chapter.

2.5. Results and Discussions

We divided our experiment into three sets. The first aims at investigating the performance of three variants of semantic tournament selection based on statistical analysis in comparison with standard tournament selection. The second attempts to improve the performance of the semantic selection strategy through its combination with a state of the art semantic crossover operator [93]. The third set of experiments examine the performance of the strategies on noisy instances of the problems. The detailed results of these experiment sets are presented in the following subsections.

[3]https://github.com/chuthihuong/GP
[4]http://www.tree-lab.org/index.php/resources-2/downloads/open-source-tools

2.5.1. Performance Analysis of Statistics Tournament Selection

This subsection analyses the performance of three statistical tournament selection methods and compares them with GP (GP with standard tournament selection) and semantic in selection (SiS) by Galvan-Lopez et al [29]. SiS is probably the most similar approach to the proposed methods.

The first metric is the mean best fitness values across 100 runs on the training data and this is presented in Table 2.3. This table shows that three new selection methods did not help to improve the performance of GP on the training data. By contrast, the training error of standard tournament selection is often significantly better than that of statistics tournament selection. This result is not very surprising since the statistics tournament selection techniques impose less pressure on the improving training error compared to standard tournament selection. For SiS, it is also slightly worse than standard tournament selection. The training error of SiS is significantly worse than that value of standard tournament selection on 5 problems with tour-size=3 and one problem with tour-size=7, while the training error of standard tournament selection is not significantly worse than that of SiS on any problems.

The second metric used in the comparison is the generalisation ability of the tested methods. In each run, the best solution was selected and evaluated on an unseen data set (the testing set). The median of these values across 100 runs was calculated and the results are shown in Table 2.4. We can see that the testing error of SiS and GP are roughly

Table 2.3: Mean of best fitness with tour-size=3 (the left) and tour-size=7 (the right).

Pro	GP	SiS	TS-R	TS-S	TS-P	GP	SiS	TS-R	TS-S	TS-P
A. **Benchmarking Problems**										
F1	2.01	$\underline{\mathbf{1.91}}$	2.74^-	2.98^-	2.70^-	$\underline{1.46}$	1.50	2.29^-	3.13^-	2.29^-
F2	0.24	$\underline{\mathbf{0.24}}$	0.39^-	0.56^-	0.31^-	0.23	$\underline{\mathbf{0.22}}$	0.35^-	0.55^-	0.26^-
F3	5.19	$\underline{\mathbf{4.94}}$	6.62^-	6.36^-	6.15^-	4.62	$\underline{\mathbf{3.62}}$	5.66^-	6.29^-	4.93^-
F4	0.05	0.06	0.05	$\underline{\mathbf{0.05}}$	**0.05**	0.05	0.05	0.05	0.05	$\underline{\mathbf{0.03}}^+$
F5	0.126	0.133^-	0.130	$\underline{\mathbf{0.126}}$	0.127	0.124	0.126	0.129	0.127	$\underline{\mathbf{0.123}}$
F6	$\underline{0.44}$	0.46	0.76^-	0.99^-	0.59^-	0.33	$\underline{\mathbf{0.31}}$	0.62^-	1.09^-	0.48^-
F7	$\underline{0.43}$	0.45	0.47^-	0.51^-	0.44	0.42	0.43	0.44	0.52^-	$\underline{\mathbf{0.40}}$
F8	4.37	$\underline{\mathbf{3.85}}$	4.43	4.72	**4.12**	5.22	**4.85**	$\underline{\mathbf{4.51}}$	5.71^-	**4.76**
F9	1.48	1.50^-	1.98	1.96	$\underline{\mathbf{1.32}}$	1.62	1.90	$\underline{\mathbf{1.42}}^+$	2.30^-	1.66
F10	7.93	8.77	7.94	$\underline{\mathbf{6.39}}$	8.78	7.72	**7.06**	**6.65**	$\underline{\mathbf{5.38}}^+$	8.51
F11	0.10	0.10	0.11	$\underline{\mathbf{0.07}}$	**0.09**	0.10	**0.08**	0.11	$\underline{\mathbf{0.07}}$	**0.10**
F12	$\underline{7.04}$	7.13^-	7.15^-	7.11^-	7.14^-	$\underline{6.96}$	7.07^-	7.14^-	7.11^-	7.06^-
F13	$\underline{0.87}$	0.87	0.89^-	0.88^-	0.88^-	0.88	0.89	0.89^-	0.89^-	$\underline{\mathbf{0.88}}^+$
F14	$\underline{76.12}$	76.54	80.58	76.72	79.68	75.84	76.75	76.26	$\underline{\mathbf{74.08}}$	76.15
F15	2.55	2.85	2.64	$\underline{\mathbf{2.45}}$	2.61	$\underline{2.17}$	2.33	2.23	2.29	2.37
B. **UCI Problems**										
F16	9.74	$\underline{\mathbf{9.13}}$	10.19	9.83	10.39	$\underline{8.04}$	8.15	8.77	8.40	8.65
F17	$\underline{3.69}$	3.75	4.05^-	4.11^-	3.97^-	$\underline{3.39}$	3.46	3.89^-	4.11^-	3.82^-
F18	$\underline{10.62}$	11.51	11.61	11.43	12.04	9.72	$\underline{\mathbf{9.07}}$	11.05	**9.41**	10.06
F19	26.43	$\underline{\mathbf{26.36}}$	29.42^-	31.63^-	28.57^-	25.25	$\underline{\mathbf{24.79}}$	29.98^-	31.88^-	27.98^-
F20	$\underline{10.50}$	10.88	10.80	10.78	10.64	$\underline{9.35}$	9.43	9.87^-	9.85^-	9.70
F21	$\underline{4.35}$	4.42^-	4.47^-	4.41^-	4.44^-	$\underline{4.23}$	4.26	4.33^-	4.35^-	4.28
F22	$\underline{1.10}$	1.24^-	1.33^-	1.29^-	1.24^-	0.84	$\underline{\mathbf{0.84}}$	1.02^-	1.22^-	0.93
F23	$\underline{4.24}$	4.36	5.35^-	4.66	5.01^-	3.47	$\underline{\mathbf{3.47}}$	4.58^-	7.22^-	4.18^-
F24	$\underline{8.99}$	9.18	10.73^-	10.91^-	10.35^-	8.08	$\underline{\mathbf{8.05}}$	10.22^-	12.14^-	9.47^-
F25	$\underline{4.98}$	5.00	6.18^-	6.69^-	5.86^-	4.47	$\underline{\mathbf{4.44}}$	5.79^-	7.18^-	5.40^-
F26	$\underline{52.00}$	52.14	52.10	52.18^-	52.07	$\underline{51.77}$	51.84	51.97	52.09^-	51.94

Table 2.4: Median of testing error with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	SiS	TS-R	TS-S	TS-P	GP	SiS	TS-R	TS-S	TS
A. **Benchmarking Problems**										
F1	8.55	9.06	**5.31**$^+$	**3.93**$^+$	**6.68**$^+$	11.3	**10.2**	**6.13**$^+$	**3.97**$^+$	**6**
F2	0.96	0.99	**0.88**	**0.81**$^+$	**0.92**	0.98	1.00	**0.89**$^+$	**0.82**$^+$	**0**
F3	33.4	34.2	**15.7** $^+$	**14.2**$^+$	**16.7** $^+$	33.4	34.2	**14.8** $^+$	**13.5**$^+$	**16**
F4	0.06	**0.06**	**0.06**	**0.05**	0.06	0.06	**0.05**	**0.05**	0.06	**0**
F5	0.135	0.137$^-$	0.136	**0.131**	**0.135**	0.135	0.135	**0.135**	**0.130** $^+$	**0.1**
F6	1.78	**1.45**	1.89	1.90	**1.63**	1.34	**1.21**	1.64	1.97	1
F7	1.70	**1.69**	1.74	**1.56**$^+$	1.75	1.77	**1.75**	1.80	**1.54**$^+$	**1**
F8	6.20	**4.52**	6.75	6.90	**5.52**	7.38	**6.79**	**6.96**	**7.38**	**6**
F9	1.66	**1.63**	**1.61**	**1.58**$^+$	**1.57**	1.71	**1.66**	**1.59**$^+$	**1.59**$^+$	**1**
F10	41.1	**38.9**	46.1	**35.8**	**39.1**	56.5	**42.2**	**38.1**	**34.1**	**52**
F11	0.085	0.086	0.086	**0.084**	0.085	0.084	0.085	0.085	**0.083**	0.0
F12	7.39	**7.33**$^+$	**7.32**$^+$	**7.32**$^+$	**7.33**	7.42	**7.41**	**7.35**$^+$	**7.33**$^+$	**7**
F13	0.876	**0.875**	**0.873**	**0.872** $^+$	**0.875**	0.878	**0.875**	**0.874**$^+$	**0.871** $^+$	**0.8**
F14	128.7	130.5	130.0	**128.4**	**126.2**	127.8	129.7	**124.7**	**122.7**	**12**
F15	4.95	5.32	5.14	**4.74**	**4.90**	4.23	4.43	4.32	**3.92**	4
B. **UCI Problems**										
F16	20.9	**20.8**	27.1	25.5	27.8	18.8	23.7	24.6	24.5	24
F17	4.99	**4.93**	**4.90**	**4.78**$^+$	**4.87**	4.93	4.99	**4.91**	**4.70**$^+$	**4**
F18	8.72	9.07	10.1	10.1	11.1	8.70	**8.31**	10.3	8.79	8
F19	41.3	**41.1**	**38.8**$^+$	**36.1**$^+$	**39.9**$^+$	42.1	42.2	**38.5**$^+$	**36.8**$^+$	**39.**
F20	9.54	9.91	9.75	9.56	9.75	9.42	9.57	9.68	9.52	9
F21	4.35	4.38	4.44	4.37	4.42	4.27	4.29	4.32	4.30	4
F22	2.11	**2.03**	2.40	**1.93**	**2.09**	1.86	**1.81**	1.97	1.90	**1**
F23	7.74	**7.24**	7.99	**5.89**$^+$	7.90	6.65	7.37	**6.63**	8.04$^-$	**6**
F24	16.7	18.4	**15.8**	**14.9**$^+$	18.1	17.5	18.3	**15.5**$^+$	**13.3**$^+$	**16**
F25	8.70	**8.44**	**8.31**	**7.99**$^+$	**8.48**	8.89	**8.43**	**7.99**$^+$	**8.40**$^+$	**8**
F26	46.05	46.14	**46.04**	**46.04**$^+$	46.14	50.33	**47.94**	**49.67**	**48.54**$^+$	**47**

equal. The difference between the testing error of two techniques is often marginal. SiS is only significant better than GP on F12 and GP is only significantly better than SiS on F5 with tour-size=3. Conversely, the testing error of three statistics tournament selection are often smaller than that of GP. Among three statistics tournament selection, the performance of TS-S is the best on the testing data. TS-S achieved the best performance on 18 problems with tour-size=3 and 14 problems with tour-size=7. This result on the testing demonstrates that, using statistical test to only select the winner individual for the mating pool when the individual is statistically better than others help to improve the generalization of GP.

In terms of statistical comparison using Friedman's test, the proposed tournament selection is often significantly better (with the confident level of 95%) than GP. For tour-size=3, TS-R is significantly better than GP on 4 problems, TS-S is significantly better than GP on 13 problems and TS-P is significantly better than GP on 3 problems. For tour-size=7, these values are 9, 13 and 6 problems, respectively. Conversely, GP is only significantly better than TS-S on one problem with the tour-size=7 (F23).

The third metric is the average size of their solutions. These values are presented in Table 2.5. While the solutions found by SiS are often as complex as those found GP, the solutions found by statistics tournament selection are simpler than those of GP and SiS. Statistical test using Friedman's test also show that the size of the solutions obtained by TS-R, TS-S and TS-P is significantly smaller than that of GP on most problems.

Table 2.5: Average of solution's size with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	SiS		TS-R		TS-S	TS-P		GP	SiS		TS-R
A. **Benchmarking Problems**												
F1	280	**276**		**244**		$\underline{\mathbf{121}}^{+}$	**238**	+	286	292		**264**
F2	169	170		**130**	+	$\underline{\mathbf{35}}^{+}$	**148**	+	160	173		**150**
F3	263	270		**262**		$\underline{\mathbf{124}}^{+}$	277		262	276		278
F4	171	190		225		$\underline{\mathbf{79}}^{+}$	197		184	**177**		217
F5	89	**78**		91		$\underline{\mathbf{53}}^{+}$	105		91	**85**		93
F6	167	**156**		**141**	+	$\underline{\mathbf{50}}^{+}$	**159**		137	151		**134**
F7	142	**138**		**128**		$\underline{\mathbf{48}}^{+}$	147		133	136		149
F8	180	**166**		**174**		$\underline{\mathbf{108}}^{+}$	184		247	**215**		**197**
F9	166	**141**		**129**	+	$\underline{\mathbf{73}}^{+}$	**140**		227	**176**	+	**161**
F10	161	**145**		162		$\underline{\mathbf{108}}^{+}$	162		184	**172**		**165**
F11	148	**139**		149		$\underline{\mathbf{76}}^{+}$	152		149	151		154
F12	259	**222**	+	**179**	+	$\underline{\mathbf{92}}^{+}$	**188**	+	306	**258**		**211**
F13	169	**146**		**119**	+	$\underline{\mathbf{32}}^{+}$	**135**	+	161	**152**		**117**
F14	254	**210**		268		$\underline{\mathbf{168}}^{+}$	273		332	**284**		**300**
F15	155	**132**		**147**		$\underline{\mathbf{112}}^{+}$	163		157	**150**		**133**
B. **UCI Problems**												
F16	200	**184**		**193**		$\underline{\mathbf{152}}^{+}$	**189**		262	**252**		**260**
F17	207	**182**		**168**		$\underline{\mathbf{50}}^{+}$	**165**		230	**220**		**192**
F18	160	**150**		165		$\underline{\mathbf{119}}^{+}$	165		226	**207**		**206**
F19	208	**200**		**119**	+	$\underline{\mathbf{16}}^{+}$	**152**	+	313	**280**		**100**
F20	198	**177**		**193**		$\underline{\mathbf{125}}^{+}$	199		299	**290**		**264**
F21	178	**149**	+	179		$\underline{\mathbf{97}}^{+}$	179		236	**196**		**219**
F22	186	**171**		**178**		$\underline{\mathbf{105}}^{+}$	**182**		194	**185**		**190**
F23	160	**159**		**132**	+	$\underline{\mathbf{56}}^{+}$	**134**	+	204	**200**		**149**
F24	164	168		**115**	+	$\underline{\mathbf{45}}^{+}$	**137**		220	**195**		**131**
F25	170	**167**		**111**	+	$\underline{\mathbf{31}}^{+}$	**138**	+	226	**205**		**132**
F26	161	**129**		**159**		$\underline{\mathbf{107}}^{+}$	**156**		249	**213**		**226**

Especially, the size of the solutions of TS-S is always much smaller than that of GP on all problems. This provides a reason partially explaining why the performance of TS-S on the testing data is better than other techniques in Table 2.4 following the Occam Razor principle [75].

We also measured the semantic distance between parents and their children of GP, SiS and TS-S[5] using the similar approach to Nguyen et al. [82]. This information shows the ability of a method to discover different areas in the search space. The semantic distance between a pair of individuals (e.g. a parent and a child) averaged over the population and over 100 runs with tour-size=3 is presented in Table 2.6.

Table 2.6: Average semantic distance with tour size=3. Bold indicates the value of SiS and TS-S is greater than the value of GP.

Pro	GP	SiS	TS-S	**Pro**	GP	SiS	TS-S
F1	2.42	**8.93**	**3.76**	F14	11.22	**11.88**	**12.30**
F2	0.42	**1.60**	**0.43**	F15	10.36	**12.28**	**12.79**
F3	7.01	**19.73**	5.02	F16	71.08	**101.43**	**78.42**
F4	1.05	**1.55**	**1.24**	F17	60.91	13.52	**136.99**
F5	0.07	**0.63**	**0.10**	F18	105.55	**366.87**	**123.34**
F6	0.99	**1.58**	**1.03**	F19	35.12	**51.21**	7.83
F7	0.44	**0.58**	**0.70**	F20	12.43	**21.87**	**15.25**
F8	38.28	**426.28**	**57.13**	F21	62.09	20.84	**96.44**
F9	8.79	**16.88**	**8.86**	F22	6.83	**7.56**	**11.61**
F10	8.16	**12.66**	**10.65**	F23	42.25	29.08	**53.12**
F11	4.32	**5.18**	**6.13**	F24	43.10	**44.05**	**80.74**
F12	6.12	**7.10**	**8.81**	F25	37.19	17.49	**41.42**
F13	2.81	**3.86**	**10.69**	F26	54.86	**56.66**	**78.18**

Apparently, both SiS and TS-S maintained higher semantic diversity

[5]We focus on analysing TS-S since this is the best selection approach among three proposed techniques.

compared to GP. TS-S and SiS preserved better semantic diversity than GP on 24 and 22 problems, respectively. These results show that TS-S achieved one of its objective in enhancing semantic diversity of GP population.

The last result in this subsection is the percentage of rejecting the null hypothesis ($N_{rejnull}$) in Wilcoxon test of TS-S and TS-R with tour-size=3. This value is calculated in Equation 2.5.

$$N_{rejnull} = \frac{N_p}{N_{test}} \tag{2.5}$$

in which N_p is the number of the Wilcoxon test rejecting the null hypothesis and N_{test} is the total number of the Wilcoxon test conducted in each selection technique. Table 2.7 shows that there is a large number of the test that rejected the null hypothesis. This value of TS-R is often from 30% to 70% and the value of TS-S is slightly higher (from 50% to nearly 90%). Thus, the statistical test will have a considerable impact

Table 2.7: Average percentage of rejecting the null hypothesis in Wilcoxon test of TS-R and TS-S with tour-size=3.

Pro	TS-R	TS-S	**Pro**	TS-R	TS-S	**Pro**	TS-R	TS-S
F1	25.32%	50.91%	F10	71.95%	81.93%	F19	39.83%	86.09%
F2	19.68%	53.97%	F11	62.18%	84.25%	F20	75.61%	84.27%
F3	30.75%	51.26%	F12	43.31%	69.85%	F21	66.13%	77.81%
F4	63.60%	74.61%	F13	38.82%	78.64%	F22	71.36%	80.85%
F5	59.65%	68.34%	F14	66.87%	78.65%	F23	43.22%	77.46%
F6	39.80%	47.88%	F15	85.17%	87.53%	F24	42.26%	73.06%
F7	33.69%	63.44%	F16	82.01%	87.30%	F25	35.80%	79.95%
F8	70.29%	88.25%	F17	43.98%	69.74%	F26	78.53%	84.77%
F9	62.91%	77.23%	F18	82.29%	87.28%			

to the selection process in TS-R and TS-S [17].

Overall, three statistics tournament selection methods find simpler solutions and generalize better on unseen data even though they do not improve the training error. Particularly, the solutions found by TS-S are much less complex than those of GP. Moreover, the generalization ability of TS-S is also better compared to GP and SiS.

2.5.2. Combining Semantic Tournament Selection with Semantic Crossover

The results in the previous subsection showed that three approaches to statistical-driven semantic tournament selection, and particularly TS-S, helped to increase the generalization ability of GP and find solutions of lower complexity. Moreover, the simplicity of the design of these techniques allows them to be easily combined with advances in GP search operators. In this subsection, we present an improvement of TS-S (the best approach among three proposed selection techniques) performance by combining this technique with a recently proposed crossover - random desired operator (RDO) [93]. In other words, we used RDO instead of standard crossover in TS-S. The resulting GP system is called statistics tournament selection with random desired operator and referred to as TS-RDO. The reason for combining RDO with TS-S is that the training error of TS-S is often worse than GP as shown in the previous subsection. Moreover, RDO has been showed to perform well on the training data [82, 93]. We predict that the coupling of these semantic selection and semantic crossover strategies in the form of TS-RDO will lead to the improved performance.

We compare TS-RDO with TS-S, neatGP (a state of the art bloat control approach) [112], RDO [93] and GP. The setting for RDO is similar to the setting in [82] in which a library of 1000 semantically different individuals with max depth of 4 was randomly generated. The result on the testing data of these methods is shown in Table 2.8. It can be seen that TS-RDO achieved the best result among five tested techniques. The testing error of TS-RDO is smallest on 13 and 15 problems with tour-size=3 and tour-size=7, respectively. Furthermore, TS-RDO is more frequently significantly better than GP compared to TS-S. TS-RDO is significantly better than GP on 17 problems with tour-size=3 and on 22 problems with tours-size=7 while these values of TS-S are only 10 and 11, respectively. RDO achieved the second best (behind TS-RDO) result. The testing error of RDO is often smaller than that of GP on all problems. This is consistent with the result in Pawlak et al. [82, 93] where RDO has been reported to perform well on unseen data.

Among five examined methods in Table 2.8, the performance of neatGP is the worst. neatGP is only significantly better than GP on 2 and 4 problems, while it is significantly worse than GP on 9 and 10 problems correspondingly with tour-size=3 and tour-size=7. This result is slightly different with the result in Trujillo et al. [112] where neatGP is showed performing equally well on unseen data compared to GP. The reason could be that the tested problems in our experiments are more difficult than those in [112]. Further research needed to examine this.

The average size of the solutions are presented in Table 2.9. The size of the solutions obtained by both neatGP and TS-S are significantly

Table 2.8: Median of testing error of TS-RDO and four other techniques with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RDO
A. **Benchmarking Problems**										
F1	8.55	12.5$^-$	**3.93**$^+$	8.91	**4.19**$^+$	11.3	12.5	**3.97**$^+$	**8.88**	**4.52**$^+$
F2	0.96	**0.84**$^+$	**0.81**$^+$	1.17$^-$	0.97	0.98	**0.84**$^+$	**0.82**$^+$	1.19$^-$	**0.96**
F3	33.4	**32.2**	**14.2**$^+$	**3.73**$^+$	**1.61**$^+$	33.4	**32.2**	**13.5**$^+$	**5.92**$^+$	**1.87**$^+$
F4	0.06	0.12$^-$	**0.05**	**0.02**$^+$	**0.02**$^+$	0.06	0.12$^-$	0.06	**0.02**$^+$	**0.02**$^+$
F5	0.135	0.135	**0.131**	0.14	0.14	0.135	0.135	**0.131**	0.14	0.14$^-$
F6	1.78	**1.74**	1.90	**0.00**$^+$	**0.00**$^+$	1.34	1.74$^-$	1.97	**0.00**$^+$	**0.00**$^+$
F7	1.69	**1.61**	**1.56**	**1.38**$^+$	**1.06**$^+$	1.77	**1.61**	**1.54**$^+$	**1.15**$^+$	**1.33**$^+$
F8	6.20	7.41$^-$	6.90	**0.00**$^+$	**0.00**$^+$	7.38	7.41	**7.38**	**0.00**$^+$	**0.00**$^+$
F9	1.66	2.41	**1.58**	**0.01**$^+$	**0.11**$^+$	1.71	2.41	**1.59**	**0.23**$^+$	**0.23**$^+$
F10	41.1	41.0	**35.8**	**0.00**$^+$	**0.00**$^+$	56.5	**41.0**	**34.1**	**0.47**$^+$	**0.00**$^+$
F11	0.08	0.30$^-$	**0.08**	**0.00**$^+$	**0.08**$^+$	0.08	0.30$^-$	**0.08**	**0.00**$^+$	**0.08**
F12	7.39	**7.34**	**7.32**$^+$	7.49$^-$	**7.32**	7.42	**7.34**$^+$	**7.33**$^+$	**7.40**	**7.28**$^+$
F13	0.88	**0.87**	**0.87**$^+$	0.88	**0.87**	0.88	**0.87**$^+$	**0.87**$^+$	**0.88**	**0.87**$^+$
F14	128.7	131.3$^-$	**128.4**	**124.3**	**121.8**$^+$	127.8	131.3$^-$	**122.7**$^+$	**125.9**	**121.7**$^+$
F15	4.95	5.92	**4.74**	**3.24**$^+$	**3.24**$^+$	4.23	5.92$^-$	**3.92**	**3.24**$^+$	**3.24**$^+$
B. **UCI Problems**										
F16	20.9	33.7$^-$	25.5	**6.11**$^+$	**5.75**$^+$	18.8	33.7$^-$	24.5	**6.46**$^+$	**5.91**$^+$
F17	4.99	**4.95**	**4.78**$^+$	5.50$^-$	**4.85**	4.93	4.95	**4.70**$^+$	5.63$^-$	**4.74**$^+$
F18	8.72	28.49$^-$	10.18	**3.56**$^+$	**3.56**$^+$	8.70	28.49$^-$	8.79	**3.63**$^+$	**3.61**$^+$
F19	41.3	**38.3**$^+$	**36.1**$^+$	41.5	**32.3**$^+$	42.1	**38.3**$^+$	**36.8** $^+$	**39.1**$^+$	**32.2**$^+$
F20	9.54	**9.18**	9.56	12.0$^-$	11.4$^-$	9.42	**9.18**	9.52	12.2$^-$	11.4$^-$
F21	4.35	4.52$^-$	4.37	**4.19**$^+$	**4.17**$^+$	4.27	4.52$^-$	4.30	**4.24**	**4.18**$^+$
F22	2.11	3.29$^-$	**1.93**	**1.09**$^+$	**1.15**$^+$	1.86	3.29$^-$	1.90	**1.23**$^+$	**1.23**$^+$
F23	7.74	8.44	**5.89**$^+$	**5.72**	**4.32**$^+$	6.65	8.44$^-$	8.04$^-$	6.84	**4.03**$^+$
F24	16.7	17.7	**14.9**$^+$	22.2$^-$	**14.8**	17.5	17.7	**13.3**$^+$	23.3	**14.3**$^+$
F25	8.70	8.89	**7.99**$^+$	12.2$^-$	**8.13**	8.89	**8.89**	**8.40**$^+$	16.0$^-$	**7.07**$^+$
F26	46.05	47.26	**46.05**	46.75	**45.84**	50.33	**47.26**	**47.63**	**46.10**	**45.40**$^+$

Table 2.9: Average of solutions size of TS-RDO and four other techniques with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S
A. **Benchmarking Problems**								
F1	280	**124** $^+$	**121** $^+$	**238** $^+$	**78**$^+$	286	**124** $^+$	**100** $^+$
F2	169	**60** $^+$	**35**$^+$	174 $^-$	**80** $^+$	160	**60** $^+$	**37**$^+$
F3	263	**112** $^+$	**124** $^+$	**153** $^+$	**59**$^+$	262	**112** $^+$	**98** $^+$
F4	171	**60**$^+$	**79** $^+$	320 $^-$	207 $^-$	184	**60**$^+$	**80** $^+$
F5	89	**12**$^+$	**53** $^+$	**49** $^+$	**23** $^+$	91	**12**$^+$	**42** $^+$
F6	167	**45** $^+$	**50** $^+$	**40** $^+$	**20**$^+$	137	**45** $^+$	**37** $^+$
F7	142	**50** $^+$	**48**$^+$	240 $^-$	**83** $^+$	133	**50** $^+$	**38**$^+$
F8	180	**118** $^+$	**108** $^+$	**21** $^+$	**12**$^+$	247	**118** $^+$	**101** $^+$
F9	166	**62** $^+$	**73** $^+$	**53** $^+$	**36**$^+$	227	**62** $^+$	**74** $^+$
F10	161	**60** $^+$	**108** $^+$	**71** $^+$	**51**$^+$	184	**60** $^+$	**101** $^+$
F11	148	**44** $^+$	**76** $^+$	**28** $^+$	**15**$^+$	149	**44** $^+$	**66** $^+$
F12	259	**67** $^+$	**92** $^+$	**181** $^+$	**55**$^+$	306	**67** $^+$	**89** $^+$
F13	169	**49** $^+$	**32** $^+$	**142** $^+$	**22**$^+$	161	**49** $^+$	**29** $^+$
F14	254	**66**$^+$	**168** $^+$	**160** $^+$	**72** $^+$	332	**66** $^+$	**164** $^+$
F15	155	**58** $^+$	**112** $^+$	**53** $^+$	**40**$^+$	157	**58** $^+$	**84** $^+$
B. **UCI Problems**								
F16	200	**103**$^+$	**152** $^+$	279 $^-$	**186** $^+$	262	**103**$^+$	**203** $^+$
F17	207	**62** $^+$	**50**$^+$	**207** $^+$	**106** $^+$	230	**62** $^+$	**39**$^+$
F18	160	**71**$^+$	**119** $^+$	305 $^-$	196 $^-$	226	**71**$^+$	**132** $^+$
F19	208	**79** $^+$	**16** $^+$	**83** $^+$	**9**$^+$	313	**79** $^+$	**11** $^+$
F20	198	**87**$^+$	**125** $^+$	328 $^-$	237 $^-$	299	**87**$^+$	**178** $^+$
F21	178	**63**$^+$	**97** $^+$	199 $^-$	**113** $^+$	236	**63**$^+$	**71** $^+$
F22	186	**83** $^+$	**105** $^+$	**139** $^+$	**58**$^+$	194	**83** $^+$	**85** $^+$
F23	160	**55**$^+$	**56** $^+$	245 $^-$	**118** $^+$	204	**55** $^+$	**24** $^+$
F24	164	**68** $^+$	**45**$^+$	240 $^-$	**97** $^+$	220	**68** $^+$	**20**$^+$
F25	170	**63** $^+$	**31**$^+$	227 $^-$	**92** $^+$	226	**63** $^+$	**22**$^+$
F26	161	**40**$^+$	**107** $^+$	**70** $^+$	**50** $^+$	249	**40** $^+$	**107** $^+$

smaller than that of GP. Comparing between neatGP and TS-S, the table shows that their solution's size is roughly equal. For RDO, its solutions are also often smaller than the solutions of GP. However, this seems only true on the benchmarking problems. On most of UCI problems, RDO's solutions are more complex than the solutions of GP. The best technique in Table 2.9 is TS-RDO. This method achieved the best result on most problems regarding to the solution's size. The average size of the solutions found by TS-RDO is the smallest one on 12 and 14 problems with tour-size=3 and tour-size=7, respectively.

Overall, TS-RDO improves the testing error, and further reduces the size of the solutions compared to TS-S. Moreover, this technique performs better than both RDO and neatGP, two recently proposed methods for improving GP performance and reducing GP code bloat.

2.5.3. Performance Analysis on The Noisy Data

This subsection investigates the performance of five methods in Subsection 2.5.2 on the noisy data. In data mining, it has been observed that the problems will become harder when they are incorporated with noise [101, 102]. We created a noisy dataset from the original one by adding 10% Gaussian noise with zero mean and one standard deviation in the all features and objective function of the problems in Table 2.1. Moreover, the noise is installed for both the training and the testing data. The testing error on the noisy data is shown in Table 2.10.

There are some interesting results observed from Table 2.10. First, TS-RDO performs slightly more consist on the noisy data compared to

Table 2.10: Median of testing error on the noisy data with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RD
A. **Benchmarking Problems**										
F1	9.68	13.1^{-}	**5.88**$^{+}$	10.3	**7.99**	9.19	13.1^{-}	**5.13**$^{+}$	10.2	**6.53**
F2	0.92	**0.84**	**0.81**	1.17^{-}	1.01^{-}	0.90	**0.84**	**0.79**$^{+}$	1.14^{-}	0.92
F3	29.6	32.2	**15.9**$^{+}$	**7.06**$^{+}$	**6.28**$^{+}$	34.8	**32.2**	**16.8**$^{+}$	**7.30**$^{+}$	**6.28**$^{+}$
F4	0.15	0.19^{-}	**0.145**	**0.143**	**0.136**$^{+}$	0.151	0.19^{-}	**0.147**	**0.143**	**0.138**
F5	0.14	0.14	**0.139**$^{+}$	0.141^{-}	0.14	0.14	0.14	**0.137**$^{+}$	0.14	0.14
F6	2.14	2.19	**2.10**	4.03^{-}	**1.36**$^{+}$	2.22	**2.19**	**2.07**$^{+}$	2.71	**1.39**$^{+}$
F7	1.74	**1.73**	**1.71**	**1.64**	**1.61**	1.79	**1.73**	**1.64**$^{+}$	**1.78**	**1.51**$^{+}$
F8	66.9	66.9	**66.8**$^{+}$	68.3^{-}	**66.8**$^{+}$	67.1	**66.9**	**66.8**$^{+}$	68.0	**66.6**$^{+}$
F9	5.52	5.68	**5.34**	**5.19**	**5.04**	5.49	5.68	**5.24**	**4.98**$^{+}$	**5.10**
F10	63.8	**56.4**	**56.2**	**49.2**$^{+}$	**46.3**$^{+}$	57.0	**56.4**	**55.0**	**49.6**$^{+}$	**47.1**$^{+}$
F11	0.20	0.32^{-}	**0.20**	0.20	**0.20**$^{+}$	0.20	0.32^{-}	**0.20**	**0.20**$^{+}$	**0.20**$^{+}$
F12	7.40	7.41	**7.31**$^{+}$	7.54^{-}	**7.36**	7.44	**7.41**$^{+}$	**7.34**$^{+}$	7.44	**7.30**$^{+}$
F13	0.90	**0.90**	**0.90**$^{+}$	0.91	**0.90**	0.90	**0.90**	**0.90**$^{+}$	0.90	**0.90**
F14	122.8	128.8^{-}	**122.6**	**122.8**	**122.6**$^{+}$	122.8	128.8^{-}	**122.6**	**122.8**	**122.5**
F15	5.01	6.21^{-}	5.07	**4.15**$^{+}$	**4.12**$^{+}$	4.20	6.21^{-}	**4.13**	**4.13**$^{+}$	**4.12**$^{+}$
B. **UCI Problems**										
F16	36.4	**36.3**	37.2	**12.0**$^{+}$	**11.4**$^{+}$	34.5	36.3	37.5	**12.1**$^{+}$	**11.55**
F17	5.61	**5.45**	**5.42**$^{+}$	6.41^{-}	**5.34**$^{+}$	5.70	**5.45**	**5.30**$^{+}$	6.55^{-}	**5.26**$^{+}$
F18	48.3	52.9^{-}	**46.6**	**37.8**$^{+}$	**36.8**$^{+}$	48.0	52.9^{-}	**46.3**	**38.6**$^{+}$	**36.7**$^{+}$
F19	42.8	**40.2**$^{+}$	**37.7**$^{+}$	**39.5**$^{+}$	**35.6**$^{+}$	44.9	**40.2**$^{+}$	**37.7**$^{+}$	**38.4**$^{+}$	**35.6**$^{+}$
F20	9.22	**8.72**$^{+}$	**9.13**	11.1^{-}	10.5^{-}	9.33	**8.72**$^{+}$	**9.16**	11.7^{-}	10.3^{-}
F21	4.53	4.67^{-}	4.54	**4.36**$^{+}$	**4.32**$^{+}$	4.49	4.67^{-}	4.50	**4.42**	**4.35**$^{+}$
F22	5.91	6.19^{-}	**5.80**	5.97	**5.43**$^{+}$	5.87	6.19^{-}	5.91	5.93	**5.50**$^{+}$
F23	8.84	9.15	**5.81**$^{+}$	8.96	**6.07**$^{+}$	7.52	9.15^{-}	9.19^{-}	9.29^{-}	**6.01**$^{+}$
F24	20.2	**19.1**	**17.1**$^{+}$	23.7	**16.5**$^{+}$	22.7	**19.1**	**16.9**$^{+}$	29.1	**16.0**$^{+}$
F25	9.36	9.42	**8.38**$^{+}$	14.8^{-}	**8.00**$^{+}$	9.58	**9.42**	**8.50**$^{+}$	13.9^{-}	**7.38**$^{+}$
F26	46.25	46.58	**46.23**	**46.26**	**46.16**	46.83	**46.58**	**46.61**	**46.72**	**46.62**

the noiseless data. The best testing error is mostly achieved by TS-RDO on all problems with both values of the tournament size. Second, the performance of RDO on the noisy data is not as good as on the noiseless data. This technique only achieved the best performance on one problem with tour-size=7. This evidenced that RDO is prone to be over-fitted on the noisy problems. Third, the performance of TS-S is also robust and more consistent than on the noiseless data. This method is significantly better than GP on 11 and 13 problems with tour-size=3 and tour-size=7 while these values on the noiseless data are only 10 and 11, respectively. Last, neatGP is still the worst method regarding to the generalization ability.

Since the solution's size of the tested methods on the noisy data is similar to the result on the noiseless data, it is shown in the appendix. Alternately, Figure 2.1 presents the testing error and the population size on four typical problems over the evolutionary process [6]. It can be seen that TS-RDO often achieved the lowest testing error over the whole evolution process. TS-RDO quickly achieved a good testing error and often kept improving this value. On one problem, F25, the testing error of TS-RDO slightly goes up at the last generations. However, it does not go as high as that of RDO. The second best technique is often RDO. However, this crossover is over-fitted after few generations particularly on the UCI problems like F22 and F25. The figure also shows that the testing error of GP and neatGP are usually much higher than others. Last, TS-S is the technique that has less over-fitted impact compared to

[6]The figures for other problems are presented in the supplement of the dissertation at https://github.com/chuthihuong/GP.

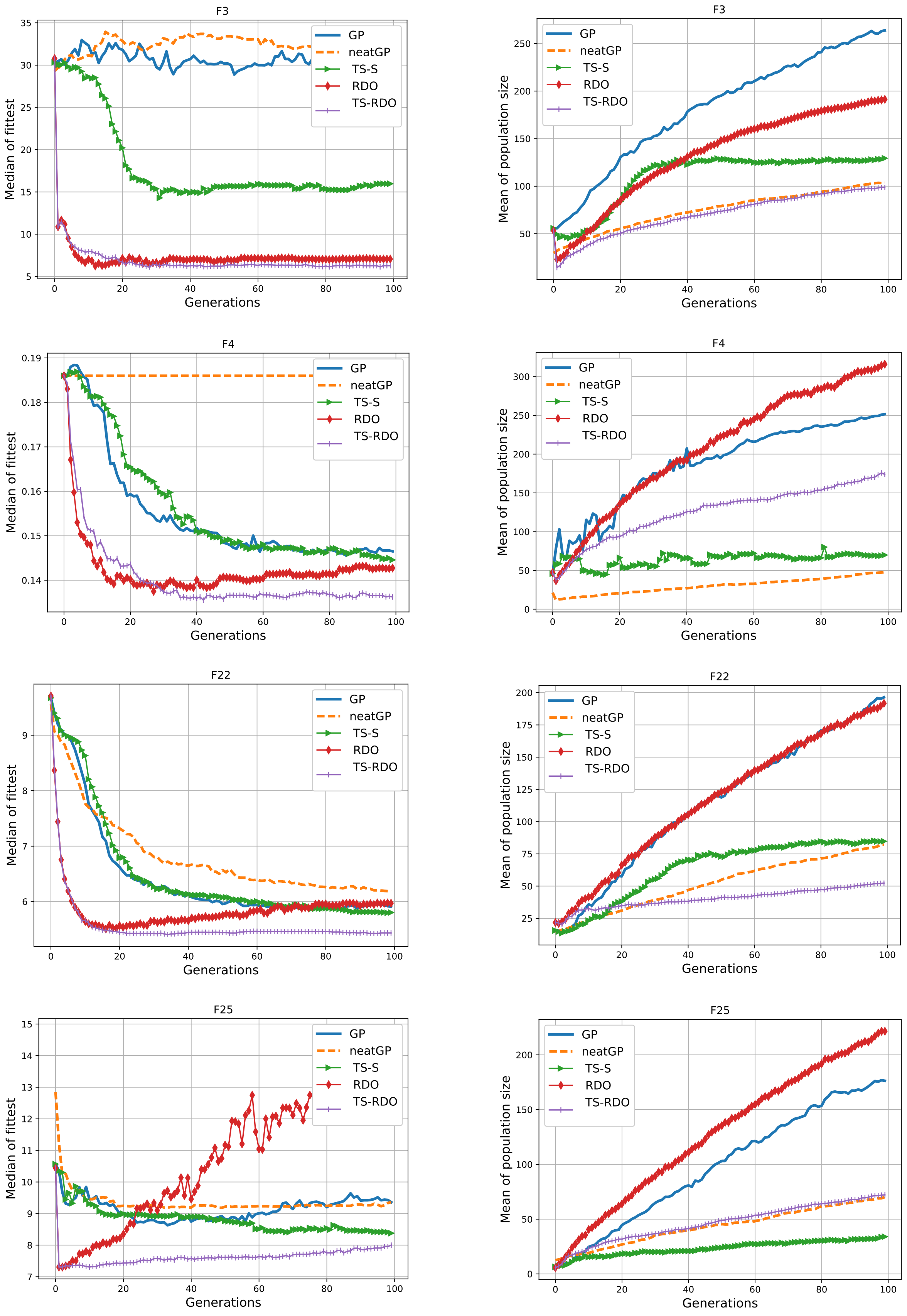


Figure 2.1: Testing error and Population size over the generations with tour-size=3.

others. This selection is only the technique that the trend of the testing error is mostly downward.

In terms of the growth of the population size, three methods including neatGP, TS-S and TS-RDO do not incur much code growth in GP population. The population size of these techniques is only slightly increased during the course of the evolution. Conversely, the population size of GP and RDO is quickly grown and it is much higher than that of neatGP, TS-S and TS-RDO.

The last experimental result analysed in this chapter is the average running time of the five methods. The average running time over 100 runs is shown in Table 2.11. Apparently, TS-S is often the fastest system among all tested methods especially with tour-size=3. This is not surprising since, the code growth of TS-S's population is much less than GP (Figure 2.1).

All other techniques need a longer running time compared to GP. For neatGP, since we used its implementation in Python, the computational time is larger than that of GP and others. RDO also requires a longer time to run compared to GP and this is consistent with the result in previous research [82]. Finally, although TS-RDO is slower than GP, its execution time has been considerably reduced compared to RDO. Thus, by combining TS-S with RDO, we achieved better testing error compared to all tested methods. Moreover, this technique also helps to reduce GP code growth and consequently lessens the computational expensive of RDO.

Table 2.12 presents the average execution time of the selection step

Table 2.11: Average running time in seconds on noisy data with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RDO
A. **Benchmarking Problems**										
F1	4	863$^-$	**1**$^+$	32$^-$	10$^-$	3	863$^-$	**2**$^+$	34$^-$	9$^-$
F2	3	501$^-$	**1**$^+$	29$^-$	10$^-$	2	501$^-$	**1**$^+$	32$^-$	10$^-$
F3	4	831$^-$	**1**$^+$	29$^-$	11$^-$	2	831$^-$	**2**	32$^-$	11$^-$
F4	12	686$^-$	6$^+$	160$^-$	117$^-$	9	686$^-$	8	144$^-$	116$^-$
F5	20	177$^-$	**18**	690$^-$	556$^-$	24	177$^-$	25	627$^-$	595$^-$
F6	3	522$^-$	**1**$^+$	104$^-$	85$^-$	17	522$^-$	**2**$^+$	142$^-$	86$^-$
F7	4	448$^-$	**2**$^+$	75$^-$	34$^-$	3	448$^-$	3	93$^-$	37$^-$
F8	51	2439	**35**$^+$	1647	1240	69	2439	72	1674	1259
F9	49	584$^-$	**36**	1762	1430	53	584$^-$	71	1755	1556
F10	67	710$^-$	**59**	1937	1536	67	710$^-$	90	1856	1738
F11	69	573$^-$	**59**	2095	1239	72	573$^-$	87	1924	1257
F12	88	678$^-$	**62**	1558	1118	91	678$^-$	104	1387	1078
F13	69	396$^-$	**36**$^+$	1697	1235	69	396$^-$	**61**	1530	1259
F14	109	867$^-$	**106**	1571	1134	117	867$^-$	136	1745	1204
F15	54	672$^-$	**48**	1478	1228	50	672$^-$	74$^-$	1390	1255
B. **UCI Problems**										
F16	27	1081	32	1195	1050	39	1081	58	1602	1072
F17	9	398$^-$	**4**$^+$	137$^-$	65$^-$	8	398$^-$	**6**	192$^-$	67$^-$
F18	36	821$^-$	37	1782	1282	40	821$^-$	62$^-$	2293	1406
F19	8	706$^-$	**1**$^+$	135$^-$	57$^-$	9	706$^-$	**3**$^+$	199$^-$	58$^-$
F20	94	879$^-$	**41**$^+$	1755	1084	120	879$^-$	**66**$^+$	2280	1063
F21	58	535$^-$	**45**	1645	1436	66	535$^-$	66	2083	1512
F22	8	670$^-$	**5**$^+$	184$^-$	133$^-$	8	670$^-$	9	222$^-$	136$^-$
F23	5	365$^-$	**2**$^+$	68$^-$	31$^-$	3	365$^-$	**2**	81$^-$	31$^-$
F24	4	367$^-$	**1**$^+$	53$^-$	30$^-$	3	367$^-$	**2**	103$^-$	28$^-$
F25	4	395$^-$	**1**$^+$	53$^-$	27$^-$	4	395$^-$	**2**	78$^-$	25$^-$
F26	522	**408**	**484**	40 389	35 838	474	**408**	614	39 577	41 971

in GP, TS-S and the average execution time of a run. It can be seen that the selection step in TS-S is slower than that of GP. However, this overhead is mostly acceptable. Moreover, since TS-S helps to reduce the code growth of GP population, the overall computational time of a run of TS-S is often smaller than that of GP.

We also observed that the execution time of the selection step in TS-S is often higher on the problems with a large number of fitness cases (F8, F9, etc.) than on the problems with a small number of fitness cases (F1, F1, etc.). On the first problems group, it is possible to further reduce the

Table 2.12: Average execution time of a run (shorted as Run) and average execution time of selection step (shorted as Tour) of GP and TS-S in seconds on noisy data with tour size=3.

Pro	GP		TS-S		**Pro**	GP		TS-S	
	Run	Tour	Run	Tour		Run	Tour	Run	Tour
F1	4	0.02	**1**	0.3	F14	109	0.01	**106**	14.8
F2	3	0.02	**1**	0.3	F15	54	0.02	**48**	16.5
F3	4	0.02	**1**	0.2	F16	27	0.01	32	10.7
F4	12	0.02	**6**	0.9	F17	9	0.01	**4**	1.3
F5	20	0.01	**18**	5.4	F18	36	0.01	37	14.1
F6	3	0.01	**1**	0.2	F19	8	0.01	**1**	0.9
F7	4	0.01	**2**	0.5	F20	94	0.01	**41**	10.2
F8	51	0.02	**35**	17.0	F21	58	0.01	**45**	14.3
F9	49	0.02	**36**	17.2	F22	8	0.01	**5**	2.0
F10	67	0.01	**59**	16.2	F23	5	0.02	**2**	0.6
F11	69	0.01	**59**	15.8	F24	4	0.02	**1**	0.6
F12	88	0.02	**62**	16.1	F25	4	0.02	**1**	0.6
F13	69	0.02	**36**	16.8	F26	522	0.04	**484**	278.6

computational time of TS-S by conducting the statistical test on only a subset of the fitness cases. An extra experiment was conducted to examine this hypothesis by applying the Wilcoxon test on 100 random values of fitness cases on the problems where the number of fitness cases is greater than 100. The results of this experiment (Table 2.13) show that this technique (TS-S-100) helps to further reduce the running time of TS-S (about 50%) while its testing error is mostly preserved. As a result, the average running time of TS-S-100 is always much smaller than that of GP.

Table 2.13: Median of testing error and average running time in seconds on noisy data with tour-size=3 when the statistical test is conducted on 100 fitness cases. The left is the median of the testing error and the right is the average running time.

Pro	F.cases	GP	TS-S	TS-S-100	GP	TS-S	TS-S-100
F5	500	0.14	**0.14**$^+$	**0.14**	20.40	**17.67**	**10.41**$^+$
F8	1000	66.95	**66.84**$^+$	**66.81**$^+$	51.42	**35.03**	**9.11**$^+$
F9	1000	5.52	**5.34**$^+$	**5.33**	48.53	**36.19**$^+$	**11.09**$^+$
F10	1000	63.89	**56.21**$^+$	**55.04**$^+$	66.99	**59.42**$^+$	**31.00**$^+$
F11	1000	0.199	**0.198**$^+$	**0.198**	68.84	**58.67**$^+$	**32.63**$^+$
F12	1000	7.4	**7.31**$^+$	**7.25**$^+$	87.78	**61.51**$^+$	**26.29**$^+$
F13	1000	0.901	**0.896**$^+$	**0.895**$^+$	69.03	**35.92**$^+$	**14.90**$^+$
F14	1000	122.86	**122.67**	**122.59**	108.65	**106.48**	**69.57**$^+$
F15	1000	5.01	5.07$^-$	**4.13**	54.01	**48.16**	**18.70**$^+$
F16	800	36.44	37.20$^-$	55.82$^-$	27.33	32.30	**18.02**$^+$
F18	1000	48.33	**46.60**$^+$	**47.49**	36.02	36.75	**16.57**$^+$
F20	750	9.22	**9.13**	**8.80**$^+$	93.63	**41.50**	**23.61**$^+$
F21	1000	4.53	4.54$^-$	4.67$^-$	58.21	**45.36**	**17.06**$^+$
F22	160	5.91	**5.80**$^+$	**5.80**	7.63	**4.89**	**3.34**$^+$
F26	5000	46.25	**46.23**	52.65$^-$	522.26	**484.13** $^+$	**17.01**$^+$

2.6. Conclusion

In this chapter, we introduced the idea of using a statistical test as part of an approach to semantic selection, which utilizes the error vectors of GP individuals. We proposed three variations of tournament selection that employ statistical analysis of these semantic vectors to select the winner for the mating pool. The proposed techniques aim at enhancing the semantic diversity and reducing the code bloat in GP population. The effectiveness of the approach was examined on a large number of symbolic regression problems including GP benchmark problems and additional problem instances drawn from UCI dataset. In the experimental results we observed that, the proposed techniques especially TS-S was better than standard tournament selection and neatGP (the state of the art method for controlling GP code bloat) in improving GP generalisation and reducing GP code growth.

One of the advantages of the proposed method is its simplicity in design and implementation. This allows it to be further improved by combining it with advanced techniques in GP. In this chapter, the best approach to semantic tournament selection, TS-S, was combined with the recently proposed semantic crossover, RDO. The resulting method, TS-RDO, achieved better testing error and reduced code growth to a greater extent. In addition, noisy instances of each problem were generated and performance of the various strategies examined demonstrating that the proposed methods have a good ability to perform well on noisy problems.

Chapter 3
SEMANTIC APPROXIMATION FOR REDUCING CODE BLOAT

Code bloat is a phenomenon in Genetic Programming (GP) characterized by the increase in individual size during the evolutionary process without a corresponding improvement in fitness. Bloat negatively affects GP performance, since large individuals are more time consuming to evaluate and harder to interpret. In this chapter, we propose a semantic approximation technique allowing to grow a (sub)tree that is semantically approximate to a given target semantics. Based on that, two approaches for reducing GP code bloat are introduced. The bloat control methods are tested on regression and time series forecasting problems. The experimental results showed that our methods help to significantly reduce code bloat and improve the performance of GP. The results in this chapter have been introduced in [C2, C5, C6, C7].

3.1. Introduction

GP has been successfully applied to many real-world problems; however, it is still not widely accepted as other machine learning approaches e.g. Support Vector Machines or Linear Regression [65]. This is due to some important shortcomings of GP such as its poor local structure, ill-defined fitness landscape and code bloat [88]. Among them, bloat phenomenon is one of the most studied problems. Bloat happens when individuals grow too large without a corresponding improvement in fitness [99, 119]. Bloat causes several problems to GP: the evolutionary process is more time consuming, it is harder to interpret the solutions, and the larger solutions are prone to overfitting. To date, many techniques have been proposed to address bloat. These techniques range from limiting individual size to designing specific genetic operators for GP [21, 53, 60, 63, 93, 104, 105, 106, 107].

In recent years, many studies have shown that the integration of semantic information into GP is highly effective and efficient. In this chapter, we propose a new usage of semantics in GP. Specifically, a technique for generating a new tree that is semantically approximate (similar) to the target semantics is introduced and used for reducing code bloat in some different strategies.

In this chapter, we present a novel approach to control GP bloat by using semantic information. The main contributions of this chapter are:

- This chapter demonstrates how the idea of semantic approximation can be utilized to reduce code growth in GP. A new proposed technique for reducing code bloat called *Semantic Approximation Technique* (SAT) is introduced. SAT allows to grow a small (sub)tree of similar semantics to a target semantic vector.

- Using SAT, two approaches for lessening GP code bloat are proposed. The first method is *Subtree Approximation* (SA) in which a random subtree is selected in an individual and this subtree is replaced by a small tree of approximate semantics. The second method is *Desired Approximation* (DA) where a new tree is grown to approximate the desired semantics of the subtree instead of its semantics.

- The performance of the bloat control strategies is examined and compared with a number of GP and non-GP systems on a large set of regression problems employing the benchmark and UCI problems. We observe that the new proposed methods help to significantly reduce code growth and improve the performance of the evolved solutions when compared to the tested methods.
- Some variants of SAT are also introduced. Base on that, several other methods for reducing GP code bloat are proposed. Furthermore, all proposed bloat control methods are intensively experimented on a real-world time series forecasting problem with difference GP parameters. The results show that these methods also robust with time series forecasting problems.

The remainder of this chapter is organised as follows. In the next section, we review the related work on managing code bloat in GP. The semantic approximation technique and some strategies for lessening code bloat are presented in Section 3.3. Then, Section 3.4 presents the experimental settings adopted in the chapter. After that, Section 3.5 analyses and compares the performance of the proposed strategies with standard GP and some recent related methods. Several properties of our proposed methods are analysed in Section 3.6. Section 3.7 compares the proposed methods with four popular machine learning algorithms. Section 3.8

applies and analyses the proposed methods on a real-world time series forecasting problem. Finally, Section 3.9 concludes the chapter.

3.2. Controlling GP Code Bloat

Due to the negative impact of code bloat, many approaches have been proposed to control bloat and lessen its impact on GP performance. Generally, the bloat control methods can be divided into three main groups: constraining individual size, adjusting selection techniques and designing genetic operators.

3.2.1. Constraining Individual Size

Earlier techniques are often based on limiting the size of individuals to prevent bloat [50, 63]. Any offspring with the size or depth above the limits is rejected and replaced by one of its parents. Some later research used the dynamic limits that are derived from the size of the best-so-far individual [104] or from the average size of the population [95, 96]. Overall, these approaches have relatively succeeded in controlling code bloat. However, it is difficult to select an appropriate value of the limits. If the limits are set at too small values, the improvement of fitness values may be prevented.

3.2.2. Adjusting Selection Techniques

The second approach is to use more than one fitness value in the selection. The parsimony pressure technique punishes the large individuals in the fitness function or prefers to choose smaller individuals in selection methods. The fitness is a linear combination of the individual size and its fitness value [9, 13], or is assigned a very bad fitness for large individuals in the population [95, 96].

Lexicographic Parsimony Pressure (LPP) [59] compares two random individuals in Boolean and Artificial Ant problems using two criteria: fitness and size. If two individuals have different fitness then the better fitness individual is chosen. Conversely, the smaller individual is selected. Double Tournament [58] selects individuals using two tournaments: one based on fitness and one based on size. The winners of fitness tournament go on to compete in the tournament using size to select the smaller individuals for the mating pool. Biased Multi-Objective Parsimony Pressure method (BMOPP) [89] sorts individuals into a Pareto layer based on the concept of denomination: an individual A dominates an individual B if A is as good as B in all objectives and is better than B in at least one objective. After the Pareto layers have been formed, individuals are selected using tournament selection. The individuals are compared with each other based on their fitness with probability p and based on their respective Pareto layers with probability $1 - p$.

An extension of LPP is Spatial Structure with Lexicographic Parsimonious Elitism (SS+LPE) [20]. SS+LPE implicitly controls bloat by mapping the population onto a 2D torus and defines a neighborhood relationship between individuals on this topology. At each generation, an individual is mated with one of its neighbors. The offspring is then compared with the parent using LPP to choose the winner for the next generation. In Chapter 2, we propose Statistics Tournament Selection with Size (TS-S) for reducing code bloat. To select the winner in a tournament, TS-S uses a statistical test on the error vector of the sampled individuals. For a pair of individuals, if the test shows that the individuals are different, then the individual with a better fitness value is the winner. Conversely, the individual with smaller size is chosen.

3.2.3. Designing Genetic Operators

The third approach has received a considerable attention, lately. Alfaro-Cid et al. [2] proposed Prune and Plant (PP) operator that splits an individual into two ones. A random subtree is selected and replaced by a terminal to create the first offspring. The selected subtree is also added to the population as the second offspring. Operator Equalisation (OE) [21] focuses explicitly on controlling the distribution of program sizes at each generation. OE determines a target histogram for the individuals in which the width of the bin determines the size of the programs belonging to the bin, and the height represents the number of individuals in the bin. OE then directs the population to the target distribution by accepting or rejecting each newly created individual into its corresponding bin. However, OE is computationally expensive because it requires to generate and reject many individuals that do not fit the desired target distribution.

Silva et al. [103, 106] used a Flat Target Distribution (Flat-OE) to address the shortcoming of OE. In other words, the range of the distribution remains constant throughout the search. Empirical results suggest that Flat-OE can control bloat while maintaining the quality of obtained solutions. Gardner et al. [31] extended OE by using the *cutoff point* concept and the accept–reject method to adjust the target distribution. The cutoff point is the size of the smallest individual which reaches a certain percentage of the best fitness so far. The bins are then only applied to the individuals of larger size than the cutoff point. Trujillo et al. [112] controlled code bloat by using neat-crossover operator in neatGP system. This operator first identifies the shared topological structure S_{ij} between parents, and then swaps of a single internal node between the parents in S_{ij}. In this way, offspring maintain the topological structure of their parents. Thus, the size of the resultant tree does not increase after crossover. neatGP was then extended to neatGP-LS by integrating with local search [41, 111].

Recently, using semantic information to control bloat, two mutation operators, including MORSM and MODO [28] were proposed. These operators choose randomly a subtree in a parent and replace it with a random subprogram taken from a Restricted Candidate List (RCL). RCL is built from a pre-defined library of subprograms so that the subprograms in RCL are not dominated for a given set of objectives (the semantic distance, the individual length). The difference between MORSM and MODO is in the first objective. While MORSM minimizes the semantic distance between the selected subtree and the subprograms, MODO minimizes the distance between the desired semantics of the selected subtree and the semantics of subprograms. In the next section, we will introduce the semantic approximation technique that allows to generate a new tree of semantically similar to a given target semantics. Using this proposed technique, we will propose some new genetic operators for reducing GP code bloat.

3.3. Methods

This section introduces a new proposed semantic approximation technique. Based on this technique, two approaches for reducing GP code bloat are continuously proposed next. The first approach replaces a random subtree in an individual by a smaller tree of approximate semantics. The second approach replaces a random subtree by a smaller tree that is semantically approximate to the desired semantics of this subtree.

3.3.1. Semantic Approximation

Searching for a tree that is semantically approximate or similar to a target semantic vector has been important in several researches [79, 93]. Nguyen et al. [79] searched for a pair of semantically similar subtree by repeatedly sampling from two parents. Pawlak and Krawiec [93] found a tree that is semantically closest to the desired semantics from a predefined library. In this chapter, we propose a novel approach to search for a tree of approximate semantics to the target semantics. This approach is called the *Semantic Approximation Technique* (SAT).

Let $s = (s_1, s_2, ..., s_n)$ be the target semantics, then the objective of SAT is to grow a tree in the form: $newTree = \theta \cdot sTree$ ($sTree$ is a small randomly generated tree) so that the semantics of $newTree$ is as close to s as possible. Let $q = (q_1, q_2, ...q_n)$ be the semantics of $sTree$, then the semantics of $newTree$ is $p = (\theta \cdot q_1, \theta \cdot q_2, ..., \theta \cdot q_n)$. To approximate s, we need to find θ so that the squared Euclidean distance between two vectors s and p is minimal. In other words, we need to minimize function

$f(\theta) = \sum_{i=1}^{n}(\theta \cdot q_i - s_i)^2$ with respect to θ. The quadratic function $f(\theta)$ achieves the minimal value at the vertex, θ^* calculated in Equation 3.1:

$$\theta^* = \frac{\sum_{i=1}^{n} q_i s_i}{\sum_{i=1}^{n} q_i^2} \tag{3.1}$$

After finding θ^*, $newTree = \theta^* \cdot sTree$ is grown, and this tree is called the approximate tree of the semantic vector s.

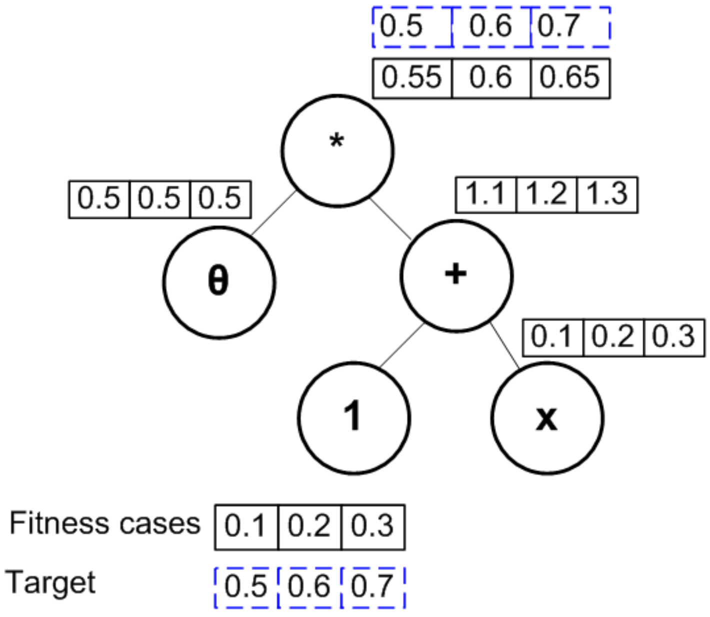


Figure 3.1: An example of Semantic Approximation

Figure 3.1 presents an example of growing a tree in the form: $newTree = \theta \cdot (1 + X)$ using SAT. In this figure, $X = \{0.1, 0.2, 0.3\}$ is the fitness cases of the problem, $s = (0.5, 0.6, 0.7)$ is the target semantics, and $\theta^* \approx 0.5$.

Compared to the previous techniques [79, 93], SAT has some advantages. First, SAT is easier to implement and faster to execute than the techniques in [79, 93]. Subsequently, GP systems based on SAT will run faster. Second, GP operators using SAT will create the population of higher diversity than the operators that only search for the subtree in other individuals [79] or from a pre-defined library [93]. Thus, SAT-based operators may achieve better performance than SSC [79] and RDO [93]. Last but not least, the size of *newTree* can be constrained by limiting the size of *sTree*, and this will be used for designing two approaches to reduce GP code bloat in the next subsections.

3.3.2. Subtree Approximation

Based on SAT, we propose two techniques for reducing code bloat in GP. The first technique is called *Subtree Approximation* (shortened as SA). At each generation, after applying the genetic operators to generate the next population, $k\%$ largest individuals in the population are selected

Algorithm 6: Subtree Approximation

```
Input: Population size: $N$, Number of pruning: $k\%$.
Output: a solution of the problem.
$i \longleftarrow 0$;
$\mathbb{P}_0 \longleftarrow InitializePopulation()$;
Estimate fitness of all individuals in $\mathbb{P}_0$;
repeat
    $i \longleftarrow i + 1$;
    $\mathbb{P}'_i \longleftarrow GenerateNextPop(\mathbb{P}_{i-1})$;
    $pool$ ⟵ get $k\%$ of the largest individuals of $\mathbb{P}'_i$;
    $\mathbb{P}_i \longleftarrow \mathbb{P}'_i - pool$;
    foreach $I' \in pool$ do
        $subTree \longleftarrow RandomSubtree(I')$;
        $S \longleftarrow Semantics(subTree)$
        $newTree \longleftarrow SemanticApproximation(S)$;
        $I \longleftarrow Substitute(I', subTree, newTree)$;
        $\mathbb{P}_i \longleftarrow \mathbb{P}_i \cup I$;
    Estimate fitness of all individuals in $\mathbb{P}_i$;
until Termination condition met;
return the best-so-far individual;
```

for pruning. Next, for each selected individual, a random subtree is chosen and replaced by an approximate tree of smaller size. Algorithm 6 presents this technique in detail.

In Algorithm 6, function $InitializePopulation()$ creates an initial population using the ramped half-and-half method. Function *Generate-NextPop*$(\mathbb{P}_{i-1})$ generates the template population using genetic operators (crossover and mutation). The next step selects $k\%$ of the largest individuals in the template population $(\mathbb{P}'_i)$ and stores them in a pool. The loop after that is used to simplify the individuals in the pool.

For each individual I' in the pool, a random subtree $subTree$ in I' is selected using function $RandomSubtree(I')$, and a small tree $sTree$ is randomly generated [1]. The semantics of $subTree$ is calculated and assigned to S (S becomes the target semantics) with function $Semantics(subTree)$. A $newTree = \theta \cdot sTree$ is grown to approximate S by using the semantic approximation technique in function $SemanticApproximation(S)$. Fi-

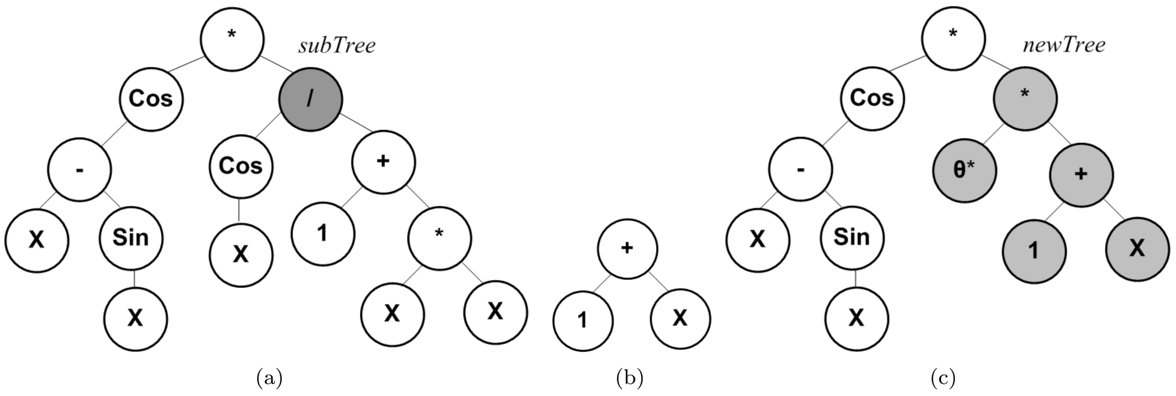


Figure 3.2: (a) the original tree with the selected subtree, (b) the small generated tree, and (c) the new tree obtained by substituting a branch of tree (a) with an approximate tree grown from the small tree (b).

[1]To reduce bloat, the size of $sTree$ needs to be smaller than the size of $subTree - 2$. If this condition is not satisfied, the process of selecting $subTree$ and generating $sTree$ is repeated. If no pair of $subTree$ and $sTree$ is found after 100 trials, individual I' is added to the next generation.

nally, individual I' is simplified to form individual I by replacing $subTree$ with $newTree$. Individual (I) is added to the next generation, $\mathbb{P}_i$. Figure 3.2 illustrates how to prune an individual by using SAT. The population at the generation i is then evaluated using the fitness function, and the whole process is repeated until the termination condition meets.

3.3.3. Desired Approximation

The second technique attempts to achieve two objectives simultaneously: lessen GP code bloat and enhance its ability to fit the training data. This technique is called *Desired Approximation* (shortened as DA).

Algorithm 7: Desired Approximation

```
Input: Population size: $N$, Number of pruning: $k\%$.
Output: a solution of the problem.
$i \longleftarrow 0$;
$\mathbb{P}_0 \longleftarrow InitializePopulation()$;
Estimate fitness of all individuals in $\mathbb{P}_0$;
repeat
    $i \longleftarrow i + 1$;
    $\mathbb{P}'_i \longleftarrow GenerateNextPop(\mathbb{P}_{i-1})$;
    $pool \longleftarrow$ get $k\%$ of the largest individuals of $\mathbb{P}'_i$;
    $\mathbb{P}_i \longleftarrow \mathbb{P}'_i - pool$;
    foreach $I' \in pool$ do
        $subTree \longleftarrow RandomSubtree(I')$;
        $D \longleftarrow DesiredSemantics(subTree)$;
        $newTree \longleftarrow SemanticApproximation(D)$;
        $I \longleftarrow Substitute(I', subTree, newTree)$;
        $\mathbb{P}_i \longleftarrow \mathbb{P}_i \cup I$;
    Estimate fitness of all individuals in $\mathbb{P}_i$;
until Termination condition met;
return the best-so-far individual;
```

DA is similar to RDO [93] in which it first calculates the desired semantics for a randomly selected subtree in an individual using the semantic backpropagation algorithm [53, 93]. However, instead of searching for a tree in a pre-defined library, DA uses SAT to grow a small tree that is semantically approximate to the desired semantics. Algorithm 7 thoroughly describes DA.

The structure of Algorithm 7 is very similar to that of SA. The main difference is in the second loop. First, the desired semantics of $subTree$ is calculated by using the semantic backpropagation algorithm instead the semantics of $subTree$. Second, $newTree$ is grown to approximate the desired semantics D of $subTree$ instead of its semantics S. By replacing $subTree$ by a $newTree$ that is semantically approximate to the desired semantics, it is predicted that the individual will be closer to the optimal value.

3.4. Experimental Settings

We tested SA and DA on twenty-six regression problems with the same dataset of Chapter 2, including fifteen GP benchmark problems recommended in the literature [120], and an additional eleven real problems are taken from UCI machine learning repository [4]. The abbreviation, the name, number of features, number of training and testing samples of each problem are presented in Table 2.1.

The GP parameters used in our experiments are typical values that are often used by GP researchers and GP practitioners [49]. They are shown in Table 3.1. The raw fitness is the root mean squared error on all fitness cases. Therefore, smaller values are better. For each problem and each parameter setting, 30 runs were performed.

Table 3.1: Evolutionary parameter values

Parameters	Value
Population size	500
Generations	100
Tournament size	3
Crossover, mutation probability	0.9; 0.1
Function set	$+, -, *, /, sin, cos$
Terminal set	$X_1, X_2, ..., X_n$
Initial Max depth	6
Max depth	17
Max depth of mutation tree	15
Raw fitness	root mean squared error on all fitness cases
Trials per treatment	30 independent runs for each value
Elitism	Copy the best individual to the next generation.

We compared SA and DA with standard GP (referred to as GP), Prune and Plant (PP) [2], TS-S and RDO [93]. PP is probably the most similar technique to SA, RDO is the inspiration for DA and TS-S is the most recently proposed bloat control method. The probability of PP operator was set to 0.5. This is the value for the best performance of PP [2, 3]. For RDO, we used a pre-defined library of 1000 subprograms with max depth of 2.

For SA and DA, 10% and 20% of the largest individuals in the population were selected for pruning. The corresponding versions were shorted as SA10, SA20, DA10 and DA20. Moreover, a dynamic version of SA (shortened as SAD) and DA (referred to as DAD) was also tested in which the individuals with the size greater than the average of the population are selected for pruning. The *newTree* was grown from *sTree* with the max depth of 2.

The source code of all tested methods are available for download[2]. All techniques were implemented in Java. Moreover, the same computing platform (Operating system: Windows 7 Ultimate (64bit), RAM 16.0GB, Intel®Core TM i7-4790 CPU@3.60GHz) was used in every experiment in this chapter.

Wilcoxon signed rank test with the confidence level of 95% is used across all the result tables in this chapter. If the test shows that a method is significantly better than GP, this result is marked + at the end. Conversely, if it is significantly worse than GP, this result is marked - at the end. Moreover, if the result of a method is better than GP, it is printed bold face, and if it is the best value, it is printed underline.

3.5. Performance Analysis

This section analyses the performance of the proposed methods using four popular metrics: training error, testing error, solution size and running time.

3.5.1. Training Error

Training error is useful for analysing the learning process of GP. Thus, it is first analysed in this section. The mean of the best fitness values in the training process across 30 runs is presented in Table 3.2.

The table shows that RDO achieved the smallest training error on

[2]https://github.com/chuthihuong/SemanticApproximation

most tested problems. This is not surprising since the main objective

Table 3.2: Mean of the best fitness

Pro	GP	RDO	PP	TS-S	SA10	SA20	SAD	DA10	DA20	DAD
A. **Benchmarking Problems**										
F1	0.47	**0.07**$^+$	1.60$^-$	0.97$^-$	0.52	0.89$^-$	1.30$^-$	**0.41**	0.97$^-$	1.17$^-$
F2	0.08	**0.02**$^+$	0.17$^-$	0.16$^-$	0.09	0.16$^-$	0.19$^-$	0.09	0.15$^-$	0.17$^-$
F3	1.91	**0.06**$^+$	4.45$^-$	**1.79**	**1.08**$^+$	2.33	4.12$^-$	**0.96**$^+$	2.2	3.58$^-$
F4	0.01	**0.00**$^+$	0.02$^-$	0.01	0.01	0.01	0.01	**0.01**	0.01	0.01
F5	0.01	**0.01**	0.01$^-$	**0.01**	**0.01**$^+$	0.01$^-$	0.01$^-$	**0.01**	0.01	0.01
F6	0.12	**0.00**$^+$	0.23$^-$	0.26$^-$	**0.09**	**0.07**$^+$	**0.06**$^+$	**0.05**$^+$	**0.03**$^+$	**0.01**$^+$
F7	0.1	**0.05**$^+$	0.15$^-$	0.15$^-$	0.12	0.14$^-$	0.12$^-$	**0.09**	0.12$^-$	0.11
F8	0.16	**0.01**$^+$	0.42$^-$	0.21$^-$	**0.00**$^+$	**0.05**$^+$	**0.06**$^+$	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$
F9	0.51	**0.05**$^+$	1.26$^-$	0.91$^-$	**0.06**$^+$	0.83	1.88$^-$	**0.13**$^+$	**0.37**	1.04$^-$
F10	1.13	**0.55**$^+$	1.87$^-$	**1.08**	**0.95**	1.75$^-$	2.64$^-$	**0.92**$^+$	1.54$^-$	2.39$^-$
F11	0.01	**0.00**$^+$	0.01$^-$	**0.00**	**0.00**$^+$	**0.00**	**0.00**	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$
F12	0.26	**0.25**	0.27$^-$	0.27$^-$	**0.25**	**0.25**	**0.26**	**0.25**	**0.25**	**0.25**
F13	0.03	**0.03**	**0.03**$^+$	0.04$^-$	**0.03**	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$
F14	9.9	**7.74**	27.02$^-$	13.10$^-$	10.45	26.50$^-$	37.12$^-$	**7.57**	17.06$^-$	26.35$^-$
F15	0.38	**0.32**	0.51$^-$	**0.37**	**0.35**	0.48$^-$	0.49$^-$	**0.35**	0.46$^-$	0.48$^-$
B. **UCI Problems**										
F16	0.41	**0.11**$^+$	1.03$^-$	**0.40**	**0.17**$^+$	**0.22**$^+$	**0.22**$^+$	**0.14**$^+$	**0.17**$^+$	**0.18**$^+$
F17	0.47	**0.39**$^+$	0.52$^-$	0.51$^-$	0.48$^-$	0.52$^-$	0.53$^-$	**0.46**	0.50$^-$	0.51$^-$
F18	0.4	**0.13**$^+$	1.32$^-$	0.42	**0.19**$^+$	**0.27**$^+$	**0.30**	**0.15**$^+$	**0.16**$^+$	**0.17**$^+$
F19	3.28	**2.97**$^+$	3.68$^-$	3.77$^-$	3.34$^-$	3.43$^-$	3.44$^-$	**3.07**$^+$	**3.26**	3.29
F20	0.51	**0.46**$^+$	0.69$^-$	0.52	0.53$^-$	0.59$^-$	0.63$^-$	0.52	0.58$^-$	0.61$^-$
F21	0.17	**0.17**$^+$	0.18$^-$	0.17	**0.17**	0.18$^-$	0.18$^-$	**0.17**$^+$	**0.17**	0.17
F22	0.17	**0.08**$^+$	0.48$^-$	0.23$^-$	**0.12**$^+$	0.53$^-$	0.62$^-$	**0.14**	0.25$^-$	0.43$^-$
F23	0.82	**0.22**$^+$	1.20$^-$	0.94	**0.65**$^+$	0.87	0.98$^-$	**0.45**$^+$	**0.52**$^+$	**0.57**$^+$
F24	1.68	**0.88**$^+$	2.05$^-$	1.93$^-$	1.7	1.99$^-$	2.05$^-$	**1.51**$^+$	1.83$^-$	1.93$^-$
F25	0.91	**0.56**$^+$	1.19$^-$	1.13$^-$	**0.90**	1.11$^-$	1.11$^-$	**0.84**$^+$	1.01$^-$	1.04$^-$
F26	1.51	1.51	1.53$^-$	**1.50**$^+$	1.52	1.53$^-$	1.53$^-$	**1.51**	1.52$^-$	1.52$^-$

of RDO is to improve the training error [93]. Among four methods for reducing code bloat (PP, TS-S, SA and DA), PP is the worst. The training error of PP is always much higher than the others. TS-S also does not improve the training error compared to GP. The mean best fitness of TS-S is slightly greater than that of GP on 19 problems. This result is consistent with the result in Chapter 2.

Conversely, the training error of SA and DA is often better than that of GP, PP and TS-S. Comparing between three configurations of SA and DA, we can see that SA10 and DA10 are better than two the other configurations. The training error of DA10 is better than GP on most tested problems, and the training error of SA10 is better than that of GP on 16 problems. This result is very impressive since the previous researches showed that bloat control methods often negatively affect the ability of GP to fit the training data [112].

To understand why SA and DA improve the training error of GP while other techniques for controlling bloat are often failed to do so, we measured the percentage of the pruning operator in PP, SA and DA that generates the offspring having better fitness than their parents. This value is calculated in Equation 3.2

$$P_{better} = \frac{N_B}{N_P} \tag{3.2}$$

in which N_P is the number of the offspring that are generated by PP, SA or DA, and N_B is the number of the offspring that is better than its parent. The value across 30 runs was averaged and shown in Table 3.3.

The table shows that PP hardly generates better offspring compared

Table 3.3: Average percentage of better offspring

Pro	PP	SA10	SA20	SAD	DA10	DA20	DAD
A. **Benchmarking Problems**							
F1	10.8%	36.3%	42.3%	46.6%	40.5%	50.7%	62.3%
F2	12.3%	37.3%	32.7%	33.9%	35.7%	45.2%	46.7%
F3	14.0%	38.3%	42.5%	50.2%	43.2%	51.7%	71.0%
F4	8.1%	50.9%	51.1%	54.4%	48.8%	49.5%	47.7%
F5	12.2%	41.0%	35.1%	28.4%	38.5%	44.5%	46.8%
F6	14.2%	43.8%	45.6%	47.0%	49.7%	54.1%	52.7%
F7	14.7%	43.8%	43.2%	44.5%	43.6%	47.4%	46.6%
F8	9.0%	48.7%	64.3%	68.6%	68.4%	68.9%	79.6%
F9	14.4%	42.7%	44.3%	54.2%	45.1%	47.6%	66.6%
F10	18.2%	43.8%	51.8%	55.8%	43.2%	49.7%	59.7%
F11	13.6%	49.1%	50.8%	60.8%	54.5%	59.0%	73.6%
F12	8.7%	45.6%	53.2%	57.6%	43.0%	49.2%	66.6%
F13	8.9%	47.2%	48.7%	58.9%	40.7%	41.7%	52.0%
F14	13.9%	43.4%	44.9%	47.2%	43.1%	57.3%	65.6%
F15	10.9%	44.0%	52.2%	60.4%	45.4%	65.4%	82.4%
B. **UCI Problems**							
F16	12.6%	44.3%	47.8%	49.9%	48.5%	68.7%	80.5%
F17	10.2%	49.3%	56.2%	63.3%	44.3%	79.9%	90.6%
F18	8.2%	49.6%	55.1%	57.5%	53.4%	65.9%	74.0%
F19	9.3%	45.2%	47.2%	47.6%	50.2%	83.0%	92.5%
F20	14.4%	48.2%	53.6%	55.3%	51.9%	58.0%	64.0%
F21	10.2%	48.6%	53.2%	60.7%	51.5%	82.1%	90.9%
F22	14.8%	48.5%	49.4%	52.0%	51.4%	68.3%	81.9%
F23	9.8%	46.2%	57.0%	60.6%	51.6%	73.9%	87.5%
F24	10.5%	45.4%	56.0%	60.0%	49.1%	77.2%	88.7%
F25	11.1%	45.1%	60.4%	61.9%	48.6%	76.0%	88.4%
F26	12.4%	47.6%	42.6%	45.7%	66.5%	86.6%	92.2%

to their parents. This value of PP is only from 10% to 15%. In contrast, this value of SA and DA is much higher (from 40% to 90%). Moreover, we can see that DA generates better offspring more frequently than SA. This is reasonable since DA used the semantic backpropagation algorithm in [93] to improve its fitness. Comparing between three versions of SA and DA, we can see that, the value of SAD and DAD are better than the value of SA10, SA20, DA10 and DA20, respectively. The reason could be that the mean best fitness of SAD and DAD is often worse than that of the other versions (Table 3.2). Thus, the probability to improve the fitness of SAD and DAD by using an operator like pruning is higher than of SA10, SA20, DA10 and DA20. Overall, the result in Table 3.3 helps to partly explain for the better training error of SA and DA compared to the other bloat control methods.

3.5.2. Generalization Ability

This subsection analyses the generalization ability of the tested methods through comparing their testing error. In each run, the best solution (the individual with the smallest fitness) was selected and evaluated on the testing data (an unseen data set). The median of these values across 30 runs was calculated and shown in Table 3.4.

The table shows that all configurations of SA and DA outperform GP on the unseen data. For example, the testing error of SA10 and DA10 is smaller than that of GP on 25 and 23 problems, respectively. Similarly, SA20, SAD, DA20 and DAD are considerably better than GP on most tested functions. For the other bloat control methods, PP is

Table 3.4: Median of testing error

Pro	GP	RDO	PP	TS-S	SA10	SA20	SAD	DA10	DA20	DAD
A. **Benchmarking Problems**										
F1	1.69	3.16$^-$	1.76	**1.35**$^+$	**1.28**$^+$	**1.05**$^+$	**1.44**$^+$	$\underline{\mathbf{0.80}}^+$	**1.68**$^+$	1.95
F2	0.30	0.36$^-$	**0.25**$^+$	**0.26**$^+$	**0.27**$^+$	**0.25**$^+$	$\underline{\mathbf{0.24}}^+$	**0.28**	**0.26**$^+$	**0.26**$^+$
F3	10.17	$\underline{\mathbf{1.92}}^+$	**8.00**	**6.66**	**4.41**$^+$	**5.44**$^+$	**5.44**$^+$	**4.38**$^+$	**4.67**$^+$	**5.68**$^+$
F4	0.01	$\underline{\mathbf{0.00}}^+$	0.01$^-$	**0.01**	**0.01**	**0.01**	0.01	0.01	0.01	**0.01**
F5	0.01	0.01	0.01	**0.01**$^+$	$\underline{\mathbf{0.01}}^+$	0.01	0.01	**0.01**$^+$	0.01	0.01
F6	0.01	$\underline{\mathbf{0.00}}^+$	**0.01**	0.01	**0.00**	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$
F7	0.06	**0.05**	0.06	0.06	0.06	0.06	**0.05**	**0.05**	$\underline{\mathbf{0.05}}$	**0.05**
F8	0.23	$\underline{\mathbf{0.00}}^+$	0.27$^-$	0.27	**0.00**$^+$	**0.05**$^+$	**0.05**$^+$	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$
F9	0.31	**0.01**$^+$	2.18	0.33	**0.06**$^+$	0.73	3.44$^-$	**0.01**$^+$	$\underline{\mathbf{0.01}}^+$	1.40
F10	33.01	**8.04**	**22.14**	37.53	**20.69**	**7.23**$^+$	**10.28**$^+$	**12.84**$^+$	$\underline{\mathbf{3.92}}^+$	**6.62**$^+$
F11	0.00	$\underline{\mathbf{0.00}}^+$	0.01$^-$	**0.00**$^+$	**0.00**$^+$	**0.00**	**0.00**	**0.00**$^+$	**0.00**$^+$	**0.00**$^+$
F12	0.26	**0.26**	0.27	**0.25**	**0.25**$^+$	**0.25**$^+$	$\underline{\mathbf{0.25}}^+$	**0.25**$^+$	**0.25**$^+$	**0.25**$^+$
F13	0.03	**0.03**	$\underline{\mathbf{0.03}}^+$	**0.03**	**0.03**	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$	**0.03**$^+$
F14	45.88	**44.63**	46.1	48.36	**45.40**	**44.87**$^+$	**44.34**	46.66	$\underline{\mathbf{44.28}}$	**44.33**$^+$
F15	2.19	**2.18**	**2.18**	2.19	**2.18**	$\underline{\mathbf{2.18}}^+$	**2.18**$^+$	2.2	**2.18**	**2.18**$^+$
B. **UCI Problems**										
F16	0.75	**0.29**$^+$	1.28	0.83	**0.27**$^+$	**0.27**$^+$	**0.28**$^+$	**0.26**$^+$	$\underline{\mathbf{0.23}}^+$	**0.26**$^+$
F17	0.61	0.66$^-$	**0.57**$^+$	**0.58**$^+$	**0.60**	**0.57**$^+$	**0.58**$^+$	**0.59**	$\underline{\mathbf{0.57}}^+$	**0.57**$^+$
F18	0.36	$\underline{\mathbf{0.14}}^+$	1.60$^-$	0.45	**0.21**$^+$	**0.29**$^+$	**0.32**$^+$	**0.16**$^+$	**0.17**$^+$	**0.18**$^+$
F19	5.18	**4.62**	**4.32**$^+$	**4.42**$^+$	**3.89**$^+$	**3.91**$^+$	**3.94**$^+$	**4.03**$^+$	$\underline{\mathbf{3.75}}^+$	**3.77**$^+$
F20	0.58	0.60$^-$	**0.51**	$\underline{\mathbf{0.48}}$	**0.50**	**0.52**	**0.55**$^+$	**0.51**	**0.53**$^+$	**0.53**$^+$
F21	0.18	$\underline{\mathbf{0.17}}^+$	0.18	**0.18**	**0.17**	0.18	0.18	**0.17**$^+$	**0.17**$^+$	**0.17**$^+$
F22	0.28	$\underline{\mathbf{0.15}}^+$	0.59$^-$	0.38	**0.18**$^+$	0.61$^-$	0.76$^-$	**0.21**	0.34	0.52
F23	1.44	**1.19**	**1.30**	**1.14**$^+$	**0.65**$^+$	**0.87**$^+$	**0.99**$^+$	**0.52**$^+$	$\underline{\mathbf{0.51}}^+$	**0.53**$^+$
F24	2.69	9.69$^-$	**2.14**$^+$	**2.41**	**2.42**	**2.10**$^+$	**2.04**$^+$	**2.31**	**2.08**$^+$	$\underline{\mathbf{1.97}}^+$
F25	1.77	3.91	**1.21**$^+$	**1.34**$^+$	**1.26**$^+$	**1.13**$^+$	$\underline{\mathbf{1.13}}^+$	**1.30**$^+$	**1.30**$^+$	**1.34**$^+$
F26	1.04	**1.03**	$\underline{\mathbf{1.02}}^+$	**1.03**	**1.02**$^+$	**1.02**$^+$	**1.02**$^+$	**1.03**	**1.02**$^+$	**1.02**$^+$

roughly equal to GP, and TS-S is better than GP. RDO is also better than GP on 19 problems. However, its performance on UCI problems is less convincing.

The result of the Wilcoxon test also confirms the good generalization ability of SA and DA. SA and DA are significantly better than GP on most tested problems. For instance, SA10, SA20 and SAD are significantly better than GP on 15, 18 and 18 problems, respectively. Similarly, DA is significantly better than GP on 16, 20 and 20 problems corresponding to its three configurations. TS-S also outperforms GP. The testing error of TS-S is significantly better than that of GP on 8 problems, while GP is not significantly better than TS-S on any problems. PP is also significantly better than GP on 7 problems, but GP is significantly better than it on 5 problems. For RDO, the Wincoxon test shows that it is slightly better than GP. RDO is significantly better than GP on 10 problems, it is significantly worse on 5 problems.

Overall, the result in this subsection shows that the performance of SA and DA is more convincing on the testing data compared to the training data. Moreover, these approaches achieve the best generalization ability compared to all tested methods. Perhaps, the reason for the convincing result of SA and DA on the testing data is that these techniques obtain simple solutions (Table 3.5) and smaller fitness than the other methods.

3.5.3. Solution Size

The main objective for performing bloat control is to reduce the complexity of the solutions. To validate if SA and DA achieve this objective, we recorded the size of the final solution (the individual with the best fitness) in each GP run. These values are then averaged over 30 runs and presented in Table 3.5.

The table shows that all tested methods find the simpler solutions compared to GP. Among them, SA20, SAD and DA20, DAD often find the smallest solutions. SA10 and DA10 also remarkably reduce the complexity of the solutions but to less extent compared to the other configurations. The solution size of PP is also very small. However, this value is slightly greater than the value of SA20, SAD, DA20 and DAD. The solution size of TS-S and RDO are greater than that of SA, DA and PP. This result provide partial explanation for the good generalization ability of the tested methods, particularly SA and DA following occam's razor principle [75].

3.5.4. Computational Time

The last metric we examine in this section is the average running time of the tested systems. This result is showed in Table 3.6. It can be observed that both SA and DA run faster than GP. The average running time of SA and DA are significantly smaller than that of GP on most tested problems. PP and TS-S also run faster than GP on most tested problems. However, they are still slower than four versions of SA and DA including SA20, SAD, DA20 and DAD. Conversely, RDO is much slower compared to the other methods. This is consistent with the results in Chapter 2. Comparing between various versions of SA and DA we can see that SA20, SAD, DA20 and DAD often run faster

Table 3.5: Average size of solutions

Pro	GP	RDO	PP	TS-S	SA10	SA20	SAD	DA10	DA20	D
A. **Benchmarking Problems**										
F1	295.5	**167.7** +	**66.9** +	**135.0** +	**89.3** +	**19.9** +	**18.2** +	**79.4** +	**17.3** +	**13**
F2	171.0	**115.9** +	**28.3** +	**31.2** +	**69.8** +	**19.2** +	**22.9** +	**53.3** +	**17.9** +	**20**
F3	228.3	**115.7** +	**44.8** +	**126.7** +	**82.8** +	**23.7** +	**16.5** +	**72.5** +	**26.8** +	**13**
F4	163.5	**107.1** +	**39.2** +	**82.3** +	**102.7** +	**47.0** +	**77.5** +	**104.1** +	**55.6** +	**72**
F5	100.9	**43.7** +	**23.9** +	**62.4** +	**51.9** +	**15.0** +	**14.9** +	**52.4** +	**21.1** +	**11**
F6	152.3	**12.6** +	**33.1** +	**40.3** +	**81.7** +	**39.5** +	**31.8** +	**64.1** +	**36.9** +	**31**
F7	180.9	**93.0** +	**44.3** +	**37.9** +	**60.7** +	**36.6** +	**30.4** +	**66.6** +	**49.5** +	**29**
F8	179.9	**67.4** +	**51.5** +	**95.6** +	**49.8** +	**8.3**+	**7.0**+	**50.7** +	**10.3** +	**8**
F9	187.3	**70.2** +	**19.4** +	**84.5** +	**67.2** +	**18.4** +	**13.4** +	**52.1** +	**13.1** +	**10**
F10	162.5	**96.8** +	**21.9** +	**102.6** +	**57.5** +	**17.9** +	**10.0** +	**53.2** +	**11.6** +	**10**
F11	140.1	**35.8** +	**23.7** +	**77.4** +	**62.6** +	**22.3** +	**7.2**+	**57.7** +	**15.4** +	**8**
F12	216.9	**81.5** +	**21.4** +	**105.9** +	**83.0** +	**25.0** +	**9.9**+	**69.5** +	**18.6** +	**12**
F13	153.6	**57.4** +	**21.5** +	**46.2** +	**70.1** +	**23.0** +	**18.5** +	**72.3** +	**18.6** +	**19**
F14	161	**94.9** +	**16.1** +	**121.8** +	**64.6** +	**20.0** +	**20.1** +	**58.3** +	**16.8** +	**14**
F15	237.8	**91.0** +	**30.4** +	**169.5** +	**80.3** +	**15.6** +	**12.0** +	**68.4** +	**19.2** +	**8**
B. **UCI Problems**										
F16	196.4	**148.4** +	**21.5** +	209.6	**52.6** +	**8.8**+	**9.2**+	**63.8** +	**16.3** +	**12**
F17	192	**140.7** +	**10.2** +	**72.3** +	**60.0** +	**9.6**+	**7.2**+	**77.3** +	**17.4** +	**12**
F18	151.7	164.6	**19.9** +	151.9	**55.0** +	**14.6** +	**13.4** +	**73.7** +	**21.9** +	**13**
F19	200.8	**67.0** +	**13.3** +	**18.6** +	**23.8** +	**6.9**+	**7.2**+	**64.1** +	**14.1** +	**8**
F20	196.4	**132.3** +	**27.4** +	**136.1** +	**77.1** +	**38.6** +	**26.2** +	**80.0** +	**42.6** +	**32**
F21	170.5	**79.5** +	**9.5**+	**93.7** +	**56.8** +	**9.9**+	**7.4**+	**60.7** +	**13.6** +	**10**
F22	216.5	**73.5** +	**23.3** +	**90.7** +	**59.7** +	**14.6** +	**15.6** +	**78.0** +	**19.0** +	**16**
F23	187.4	**156.3**	**10.3** +	**48.1** +	**53.2** +	**10.3** +	**7.6**+	**69.6** +	**16.3** +	**10**
F24	192.6	**161.6**	**10.0** +	**45.8** +	**61.6** +	**11.6** +	**7.9**+	**76.6** +	**17.5** +	**15**
F25	177.5	**141.6** +	**12.0** +	**49.4** +	**62.8** +	**9.0**+	**8.1**+	**66.0** +	**19.2** +	**12**
F26	177.2	**25.8** +	**14.2** +	**130.6** +	**16.1** +	**7.0**+	**7.0**+	**29.8** +	**11.1** +	**8**

Table 3.6: Average running time in seconds

Pro	GP	RDO	PP	TS-S	SA10	SA20	SAD	DA10	DA20	DAD
A. **Benchmarking Problems**										
F1	3.6	18.7 $^-$	**1.1**$^+$	**1.3**$^+$	**1.0**$^+$	**0.7**$^+$	**0.8**$^+$	**0.9**$^+$	**0.4**$^+$	**1.0**$^+$
F2	2.7	17.5 $^-$	**1.1**$^+$	**0.7**$^+$	**1.4**$^+$	**0.6**$^+$	**0.7**$^+$	**1.3**$^+$	**0.8**$^+$	**0.5**$^+$
F3	2.7	15.9 $^-$	**0.9**$^+$	**1.6**$^+$	**1.0**$^+$	**0.6**$^+$	**1.1**$^+$	**0.9**$^+$	**0.4**$^+$	**0.8**$^+$
F4	25.5	122.5 $^-$	34.5 $^-$	**6.8**$^+$	**12.1** $^+$	**11.9** $^+$	**21.8** $^+$	**8.6**$^+$	**8.8**$^+$	**9.1**$^+$
F5	31.5	468.7 $^-$	**6.9**$^+$	**20.4** $^+$	**16.4** $^+$	**6.1**$^+$	**3.1**$^+$	**20.5** $^+$	**9.3**$^+$	**8.6**$^+$
F6	14.5	70.2 $^-$	**3.2**$^+$	**1.4**$^+$	**2.4**$^+$	**2.1**$^+$	**10.2** $^+$	**2.1**$^+$	**1.9**$^+$	**2.7**$^+$
F7	4.3	33.9 $^-$	**3.5**	**1.8**$^+$	**2.0**$^+$	**2.6**$^+$	8.8	**2.9**$^+$	**1.9**$^+$	**3.6**
F8	50.0	907.0 $^-$	**9.4**$^+$	**25.7** $^+$	**12.7** $^+$	**5.4**$^+$	**5.4**$^+$	**16.4** $^+$	**5.6**$^+$	**9.0**$^+$
F9	63.2	882.7 $^-$	**15.0** $^+$	**27.7** $^+$	**16.6** $^+$	**6.4**$^+$	**8.5**$^+$	**18.5** $^+$	**7.2**$^+$	**10.5** $^+$
F10	64.9	902.9 $^-$	**13.7** $^+$	**44.7** $^+$	**16.9** $^+$	**5.7**$^+$	**7.7**$^+$	**19.5** $^+$	**7.1**$^+$	**10.9** $^+$
F11	75.3	526.8 $^-$	**19.6** $^+$	**54.6** $^+$	**65.9**	**15.4** $^+$	**5.6**$^+$	**32.1** $^+$	**11.2** $^+$	**8.8**$^+$
F12	96.0	1475.5 $^-$	**15.9** $^+$	**55.2** $^+$	**33.4** $^+$	**11.1** $^+$	**8.1**$^+$	**30.2** $^+$	**10.0** $^+$	**10.2** $^+$
F13	77.7	773.1 $^-$	**19.4** $^+$	**31.6** $^+$	**27.4** $^+$	**11.5** $^+$	**8.5**$^+$	**32.2** $^+$	**12.5** $^+$	**11.5** $^+$
F14	79.7	936.0 $^-$	**12.9** $^+$	**58.6** $^+$	**21.1** $^+$	**6.2**$^+$	**8.2**$^+$	**26.4** $^+$	**8.6**$^+$	**11.1** $^+$
F15	82.7	1232.6 $^-$	**15.3** $^+$	**61.7** $^+$	**22.8** $^+$	**7.1**$^+$	**7.4**$^+$	**26.6** $^+$	**9.1**$^+$	**9.9**$^+$
B. **UCI Problems**										
F16	46.0	629.8 $^-$	**7.0**$^+$	55.7	**7.2**$^+$	**3.4**$^+$	**5.3**$^+$	**15.3** $^+$	**5.9**$^+$	**7.0**$^+$
F17	8.4	45.5 $^-$	**2.6**$^+$	**3.2**$^+$	**2.9**$^+$	**1.3**$^+$	**2.9**$^+$	**5.6**$^+$	**1.4**$^+$	**3.7**$^+$
F18	43.8	768.8 $^-$	**10.2** $^+$	40.4	**12.9** $^+$	**6.3**$^+$	**8.1**$^+$	**19.1** $^+$	**9.1**$^+$	**11.5** $^+$
F19	7.2	35.8 $^-$	**2.0**$^+$	**1.2**$^+$	**1.1**$^+$	**1.2**$^+$	**2.8**$^+$	**4.5**$^+$	**1.5**$^+$	**3.6**$^+$
F20	95.5	1246.8 $^-$	**39.1** $^+$	**48.0** $^+$	**35.3** $^+$	**26.5** $^+$	**15.9** $^+$	**34.6** $^+$	**22.0** $^+$	**36.9** $^+$
F21	63.1	723.9 $^-$	**10.8** $^+$	**35.2** $^+$	**17.0** $^+$	**5.9**$^+$	**7.8**$^+$	**21.3** $^+$	**6.3**$^+$	**11.1** $^+$
F22	10.6	68.5 $^-$	**2.3**$^+$	**4.9**$^+$	**3.7**$^+$	**1.3**$^+$	**2.1**$^+$	**4.4**$^+$	**1.3**$^+$	**2.4**$^+$
F23	4.1	35.2 $^-$	**0.6**$^+$	**0.9**$^+$	**1.2**$^+$	**0.8**$^+$	**1.2**$^+$	**2.9**$^+$	**1.0**$^+$	**2.2**$^+$
F24	4.0	33.8 $^-$	**0.6**$^+$	**1.0**$^+$	**1.3**$^+$	**0.5**$^+$	**1.1**$^+$	**2.8**$^+$	**0.4**$^+$	**0.8**$^+$
F25	4.0	32.4 $^-$	**0.6**$^+$	**1.0**$^+$	**1.3**$^+$	**0.5**$^+$	**1.2**$^+$	**3.0**	**0.5**$^+$	**0.9**$^+$
F26	268.1	9334.5 $^-$	**33.0** $^+$	**237.0**	**18.1** $^+$	**19.3** $^+$	**30.8** $^+$	**84.7** $^+$	**20.0** $^+$	**42.6** $^+$

than SA10 and DA10. This is because the size of the individuals in the population of SA20, SAD, DA20 and DAD are often remarkably smaller than the size of the individuals in SA10 and DA10 as shown in Table 3.5. Consequently, the computing time of fitness evaluation in SA20, SAD, DA20 and DAD is much less than that in SA10 and DA10.

Overall, the results in this section show that SA and DA improve the training error and the testing error compared to GP and the recent bloat control methods (PP and TS-S). Moreover, the solutions obtained by SA and DA are much simpler, and their average running time are much less than that of GP on most tested functions.

3.6. Bloat, Overfitting and Complexity Analysis

This section presents a deeper analysis on the properties of the tested methods using three quantitative metrics proposed in [115]: bloat, overfitting and functional complexity. Due to the space limitation, we only present the results on four typical problems (F1, F13, F17 and F25) and for two configurations (SA20 and DA20). The results on the other problems and the rest versions of SA and DA are shown in the supplement of this chapter[3].

3.6.1. Bloat Analysis

Although bloat is a well-known phenomenon, it is often difficult to quantify the bloat of a GP system. Here, we use the metric proposed by Vanneschi et al. [115] to measure the amount of bloat in a GP generation.

[3]https://github.com/chuthihuong/SemanticApproximation

The bloat value ($bloat(g)$) at generation g is calculated by Equation 3.3.

$$bloat(g) = \frac{(\bar{\delta}(g) - \bar{\delta}(0))/\bar{\delta}(0)}{(\bar{f}(0) - \bar{f}(g))/\bar{f}(0)} \tag{3.3}$$

where $\bar{\delta}(g)$ is the average size of the individuals at generation g, and $\bar{f}(g)$ is the average fitness at the generation g. Bloat value presents the relationship between the average size growth and the average fitness improvement up to generation g compared to the respective values at generation zero ($f(0)$ and $\bar{\delta}(0)$).

Using Equation 3.3, we recorded the amount of bloat in each GP run and then averaged across 30 runs. The bloat values over generations on four problems, F1, F13, F17 and F25 are presented in Figure 3.3. It can be seen that, GP and RDO are two methods that incur high amount of bloat. The bloat value of these methods considerably grows over generations. Conversely, all bloat control methods do not lead to bloat. The bloat value of PP, TS-S, SA and DA are mostly stable during the evolutionary process. This result evidences that bloat control methods achieve their objective in lessening GP code growth.

3.6.2. Overfitting Analysis

A learner is overfitting if it performs well on the training data and does not perform satisfactorily on the testing data. However, quantitatively measuring the overfitting of a learner is very challenging. In this chapter, we follow Vanneschi et al. [115] to quantify the overfitting of a GP system based on its training error and testing error: at a given generation g, if the testing error (te_err) is smaller than the training error (tr_err) or is better than the best testing error so far, there is no overfitting; otherwise

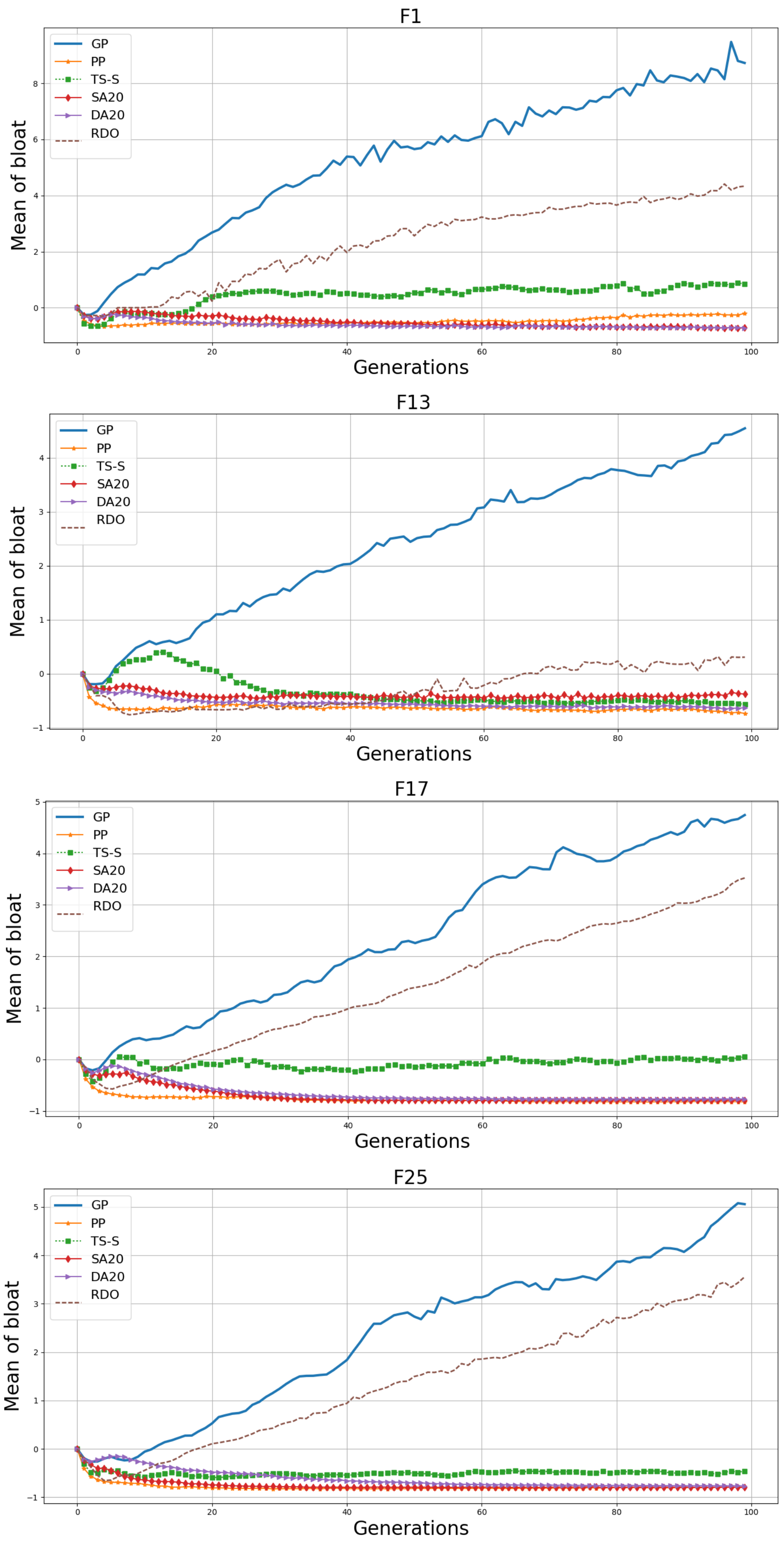


Figure 3.3: Average bloat over generations on four problems F1, F13, F17 and F25.

the overfitting is calculated by the difference of the distance between the testing error and the training error at the generation g and the distance between the testing error (te_br) and the training error (tr_br) at the generation that achieves the best testing error so far. In other words, the overfitting at generation g is calculated in Equation 3.4.

$$overfit(g) = \begin{cases} 0, & \text{if } tr\_err(g) > te\_err(g) \\ 0, & \text{if } te\_err(g) > te\_br \\ |tr\_err(g) - te\_err(g)| & \\ -|tr\_br - te\_br|, & \text{otherwise.} \end{cases} \tag{3.4}$$

Figure 3.4 shows the average overfitting over generations on four problems F1, F13, F17 and F25. The amount of overfitting of RDO is often the highest. This operator has the greatest overfitting on three out of four functions in Figure 3.4. The reason for this could be that RDO solely focuses on improving the training error but not the testing error. The second highest overfitting is GP. Although the training error of GP is not as small as RDO, the higher complexity of its solutions may be the source of overfitting. On the contrast to RDO and GP, all bloat control methods (SA, DA, TS-S and PP) do not impose much overfitting. The overfitting of these techniques are mostly stable. On one function (F13), the overfitting of SA, DA, TS-S and PP slightly increases in some early generations. However, their values are still much smaller than that of GP and RDO. For SA and DA, this result is very interesting: Although these two methods improve the training error, they do not incur the

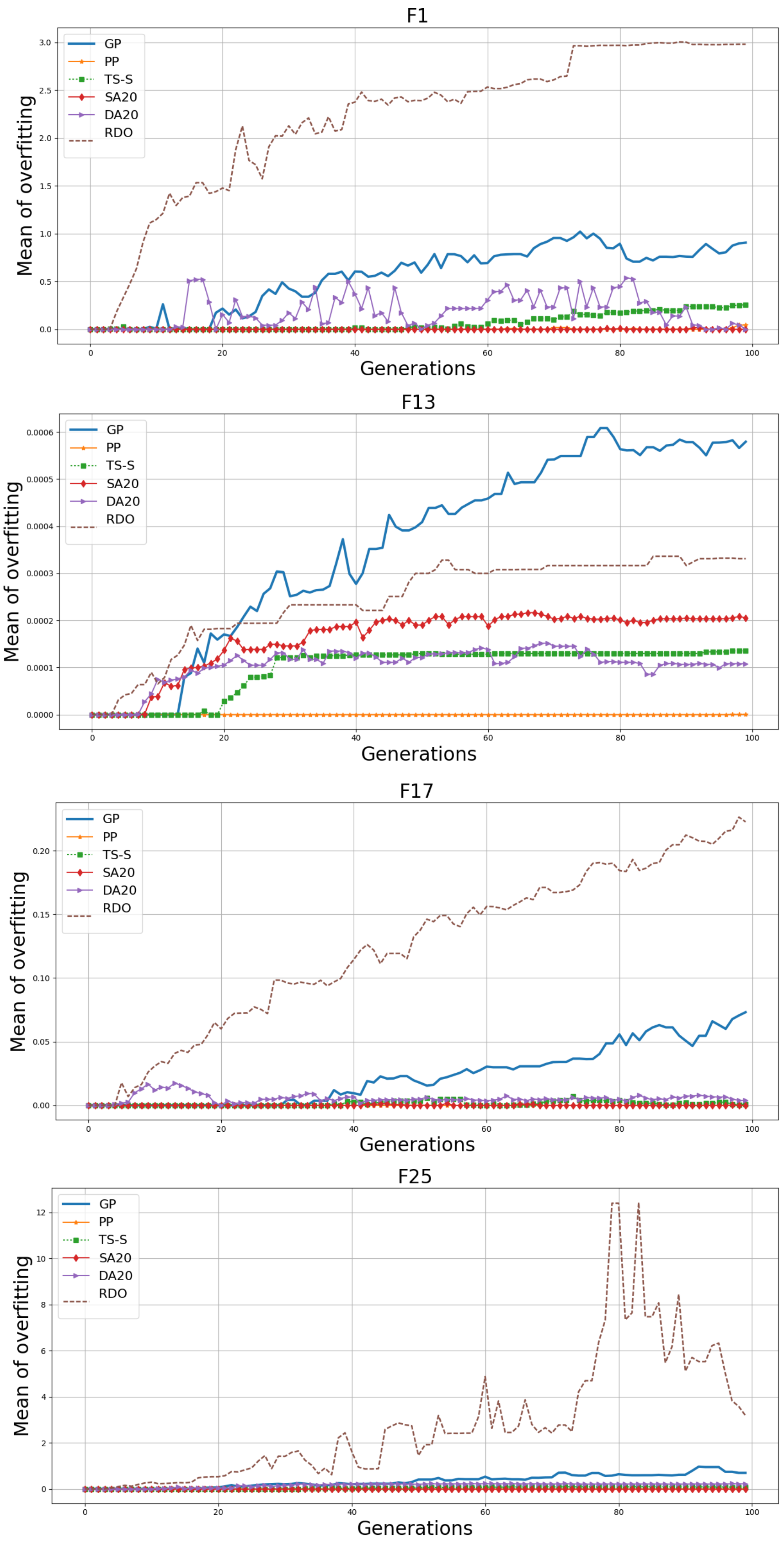


Figure 3.4: Average overfitting over the generations on four problems F1, F13, F17 and F25.

overfitting as RDO. The reason could be that the solutions obtained by SA and DA are much smaller than the solutions of RDO.

3.6.3. Function Complexity Analysis

The complexity of a GP solution is often judged based on its size (the number of nodes). However, a solution with greater size might not always be more complex than a solution of smaller size. To better quantify the complexity of GP solutions, we use a metric based on the curvature of a function [115].

Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ be the fitness cases of a problem, where each x_i is an m-dimensional vector. Let $g : \mathbf{R}^m \longrightarrow \mathbf{R}$ be the function coding a GP individual and let $g_i = g(\mathbf{x}_i)$. Let $\mathbf{p}_j = (x_{1j}, x_{2j}, ..., x_{nj})$ be the vector containing all the values of feature j in $\mathbf{X}$ and $\mathbf{q}_j = (y_{1j}, y_{2j}, ..., y_{nj})$ is a permutation of $\mathbf{p}_j$ so that the values in $\mathbf{q}_j$ are sorted in the ascending order. Let $\phi : \{1, 2, ..., n\} \longrightarrow \{1, 2, ..., n\}$ be a function that returns the position of the corresponding element of $\mathbf{q}_j$ in $\mathbf{p}_j$. Then the *partial complexity* on the j^{th} dimension is defined as:

$$pc_j = \begin{cases} \sum_{i=1}^{n-2} \left| \frac{g_{\phi(i+1)} - g_{\phi(i)}}{y_{(i+1)j} - y_{ij}} - \frac{g_{\phi(i+2)} - g_{\phi(i+1)}}{y_{(i+2)j} - y_{(i+1)j}} \right|, & \text{if } n \geq 3 \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

Finally, the complexity is calculated as the average of all the partial complexities on all the dimensions of the feature space:

$$complexity = \frac{\sum_{j=1}^{m} pc_j}{m} \tag{3.6}$$

Figure 3.5 shows the average complexity of the best individual over 30 runs across generations. Interestingly, although the solution size of

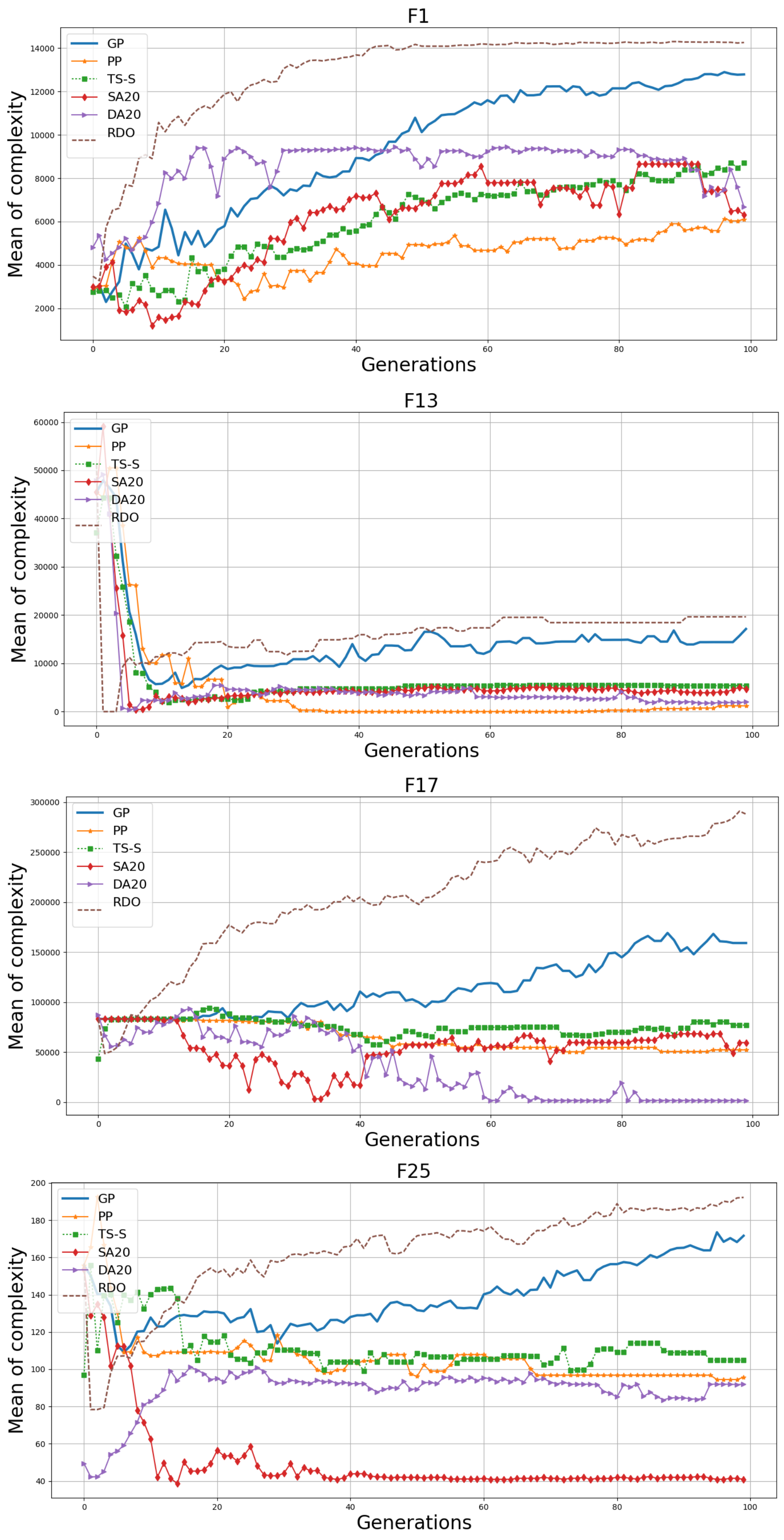


Figure 3.5: Average complexity of the best individual over the generations on four problems F1, F13, F17 and F25.

RDO is often smaller than GP (Table 3.5), the complexity of its solution is higher than GP. This partially explains for the higher overfitting value of RDO in Figure 3.4. For four bloat control methods, the figure shows that they often produce the less complex solutions. The complexity of the solutions obtained by these methods is much lower than GP and RDO. Moreover, this value does not increase much until the end of the evolution.

3.7. Comparing with Machine Learning Algorithms

This section compares the results of the proposed methods with some popular machine learning models. Four machine learning algorithms including Linear Regression (LR), Support Vector Regression (SVR), Decision Tree (DT), and Random Forest (RF) are used in this experiment. The implementation of these regression algorithms in a popular machine learning packet in Python Scikit learn [94] is used. To minimize the impact of experimental parameters to the performance of the regression algorithms, we implemented the grid search technique for tuning the important parameters for SVR, DT and RF. We did not execute the grid search for LR since it does not have parameters for tuning. The range of values for each parameter that is tuned by the grid search technique

Table 3.7: Values of the grid search for SVR, DT and RF

Methods	Parameters
SVR	'C'=[0.01, 0.1, 1, 10, 100]
DT	'max_depth'=[10, 20, 50, 100, 200]
RF	'n_estimators'=[20, 50, 100, 200] ; 'max_depth'=[10, 20, 50, 100, 200]

is presented in Table 3.7. Other parameters are used as the default settings in Scikit learn. The testing error of the proposed models and four machine learning systems are presented in Table 3.8.

It can be seen from Table 3.8 that, while standard GP is often worse than four machine learning systems, the proposed methods are competitive with machine learning algorithms. Specifically, the proposed methods achieve the smallest testing error on 12 problems while non-GP methods achieve the best result on 14 problems. Among four machine learning methods, we can see that RF is the best technique. RF achieves the best result on 6 out of 26 problems. However, the results of RF are only roughly equal to the performance of DA20. On some problems such as F9 and F10, DA20 achieves much smaller testing errors than RF and the three other machine learning techniques.

Overall, the results in this section show that our proposed methods are often better than three machine learning algorithms including LR, SVR and DT and they are as good as the best machine learning algorithm (RF) in the generalization ability. Moreover, the solution complexity of SA, DA is much simple then the solution complexity of RF that are often the combination of dozens or hundreds trees.

3.8. Applying semantic methods for time series forecasting

This section introduces some variants of SAT and proposes several bloat control methods based on that. Then, we expand the investigation of semantic methods on a real-world time series forecasting problem with different GP parameters.

Table 3.8: Comparison of the testing error of GP and machine learning systems. The best results are underlined.

Pro	GP	SA10	SA20	SAD	DA10	DA20	DAD	LR	SVR	DT	RF
A. **Benchmarking Problems**											
F1	1.69	1.28	1.05	1.44	0.80	1.68	1.95	1.85	1.64	1.50	1.45
F2	0.30	0.27	0.25	0.24	0.28	0.26	0.26	0.26	0.25	0.30	0.24
F3	10.17	4.41	5.44	5.44	4.38	4.67	5.68	6.61	5.37	7.59	5.83
F4	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01
F5	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.01	0.00
F6	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.01
F7	0.06	0.06	0.06	0.05	0.05	0.05	0.05	0.07	0.05	0.05	0.05
F8	0.23	0.00	0.05	0.05	0.00	0.00	0.00	0.00	21.77	0.12	0.06
F9	0.31	0.06	0.73	3.44	0.01	0.01	1.40	5.18	5.17	4.44	5.24
F10	33.01	20.69	7.23	10.28	12.84	3.92	6.62	81.30	81.38	81.92	82.28
F11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F12	0.26	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.30	0.26
F13	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.03
F14	45.88	45.40	44.87	44.34	46.66	44.28	44.33	44.13	43.97	45.67	51.95
F15	2.19	2.18	2.18	2.18	2.20	2.18	2.18	2.17	2.17	2.23	2.18
B. **UCI Problems**											
F16	0.75	0.27	0.27	0.28	0.26	0.23	0.26	0.22	0.32	0.31	0.23
F17	0.61	0.60	0.57	0.58	0.59	0.57	0.57	0.60	0.64	0.70	0.54
F18	0.36	0.21	0.29	0.32	0.16	0.17	0.18	0.15	0.37	0.16	0.13
F19	5.18	3.89	3.91	3.94	4.03	3.75	3.77	4.12	4.18	4.85	3.72
F20	0.58	0.50	0.52	0.55	0.51	0.53	0.53	0.50	0.49	0.47	0.40
F21	0.18	0.17	0.18	0.18	0.17	0.17	0.17	0.17	0.19	0.20	0.15
F22	0.28	0.18	0.61	0.76	0.21	0.34	0.52	0.76	1.14	0.14	0.15
F23	1.44	0.65	0.87	0.99	0.52	0.51	0.53	1.84	1.02	0.56	0.56
F24	2.69	2.42	2.10	2.04	2.31	2.08	1.97	1.83	2.47	2.53	2.04
F25	1.77	1.26	1.13	1.13	1.30	1.30	1.34	1.58	1.22	1.15	1.14
F26	1.04	1.02	1.02	1.02	1.03	1.02	1.02	1.31	1.02	3.35	1.67

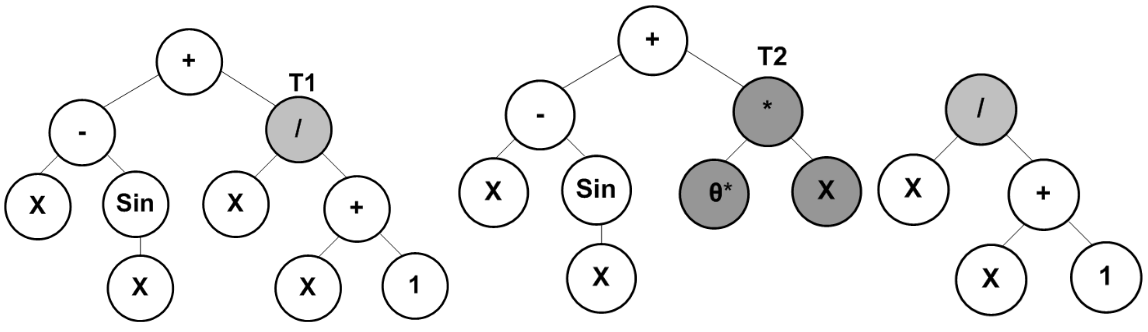
3.8.1. Some other versions

In the semantic approximation technique, an approximate tree is grown in the form: $newTree = \theta \cdot sTree$ which in $sTree$ is a small tree. Based on the growth and storage of the small tree, $sTree$, we have proposed a number of different versions of the SAT. For a generalized version, $sTree$ is a small randomly generated tree. Additionally, some of its variants can be:

- $sTree$ is a random terminal taken from the terminal set.
- $sTree$ is a small tree taken from the pre-defined library.

Based on that, we have proposed a new method called *Substituting a subtree with an Approximate Terminal* (SAT-GP) [C2]. SAT-GP uses Algorithm 6 in which $sTree$ is a random terminal that taken from the terminal set instead to randomly generate. Besides, we have also proposed an extension of SAT-GP, namely *Substituting a subtree with an Approximate Subprogram* (SAS-GP) [C5]. SAS-GP aims at having more options to select approximate trees. Thus, a pre-defined library of subprograms is used instead of the terminal set as in SAT-GP.

Moreover, the semantic approximation technique can be applied to other bloat control methods. We combine this semantic approximation technique with Prune and Plant operator [2] to create a new operator called *Prune and Plant based on Approximate Terminal* (PP-AT) [C6]. PP-AT is an extension of Prune and Plant. Figure 3.6 demonstrates an example. PP-AT selects a random subtree T_1 and then replaces it with an approximate tree T_2. T_2 is grown so that the semantics of T_2



(a) Parent (b) Child 1 (c) Child 2

Figure 3.6: An example of PP-AT.

and T_1 are most similar by using the variant of SAT where $sTree$ is a random terminal taken from the terminal set. Moreover, T_1 is grown in the population as a new other child.

3.8.2. Time series prediction model and parameter settings

The proposed semantic methods for lessening code bloat, including TS-S, SAT-GP, SAS-GP, PP-AT, SA and DA are evaluated on a real-world time series forecasting problem taken from Kaggle competition, Corporación Favorita Grocery Sales Forecasting problem [18]. The competition was opened from October 19, 2017 to January 15, 2018 on the website[4] to predict the unit sales for items sold at different Favorita stores located in Ecuador. The unit sales were daily recorded from January 2, 2013 to August 15, 2017. We downloaded a data set for one item with the identifier of 103665 at Santo Domingo city, Santo Domingo de los Tsáchilas Province, Ecuador (the identifier of this store is 5). These data are then divided into two sets, training set and testing set. The data from 1/2/2013 to 4/15/2017 are used for training, and the others are used for testing. Figure 3.7 plots the data from 9/1/2016 to

[4]https://www.kaggle.com/c/favorita-grocery-sales-forecasting

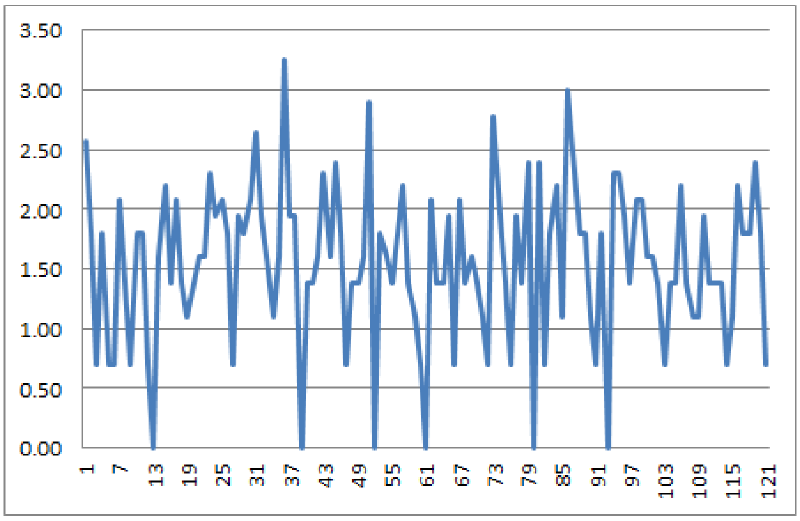


Figure 3.7: Plot of log(unit sale + 1) from 9/1/2016 to 12/31/2016.

12/31/2016. Totally, there are 1549 samples used for training and 121 samples used for testing.

Generally, there are two time series prediction models: one-step and multi-step prediction [98, 80]. The objective of one-step prediction is to express the future value of $x(t+1)$ as a function of the N previous values in the time series, $x(t), x(t-1), ..., x(t-N+1)$. That is to find the function F so that:

$$x(t+1) = F(x(t), x(t-1), ..., x(t-N+1)) \tag{3.7}$$

The task of multi-step prediction is a prediction of several steps ahead in the future, $x(t+1), x(t+2), x(t+3), ...$

In the experiments, we use the one-step prediction model and set N at 7 [5]. In other words, the main task of GP is to find a model for $x(t+1)$ based on its 7 previous values, $x(t), x(t-1), ..., x(t-6)$.

The GP hyper-parameters used for these experiments are shown in

[5]These data were collected daily, so we predict the value in the future based on the values of 7 successively previous points.

Table 3.1. Moreover, in order to investigate the code bloat control methods based on semantics, we examine three population sizes, 250, 500 and 1000, and four values of generation including 25, 50, 100 and 150.

We compare the semantic methods with standard GP (referred to as GP) and Prune and Plant (PP) [2]. The probability of PP and PP-AT is set to 0.5, and $k\%$ the largest individuals of SAT-GP, SAS-GP, SA and DA is set 10%. In SAS-GP, the pre-defined library of subprograms has 1000 subtrees with max depth of 2.

3.8.3. Results and Discussion

This subsection compares the results of the semantic methods with standard GP and PP [2]. Four metrics including the training error, the testing error, the code bloat effect and the evolution time are used to evaluate the performance of these methods.

The first metric measures the mean of best fitness on the training data, and these results are shown in Table 3.9. The table indicates that the training error of all tested GP systems normally decreases as population size increases, and the training error of the methods based on SAT, including SAT-GP, SAS-GP, SA and DA (referred to SAT-based methods hereafter) are usually better than that of GP with the population size of 250 and 500. However, when using the population size of 1000, the mean of the best fitness created from these methods are mostly worse than that of GP on all the settings of generation.

For the remaining methods, the training error of PP and PP-AT are often significantly worse than that of GP. However, compared to PP,

Table 3.9: Mean of the best fitness

Pop	Gen	GP	TS-S	PP	PP-AT	SAT-GP	SAS-GP	SA	DA
250	25	0.710	**0.705**	0.727^{-}	0.714^{-}	**0.702**$^{+}$	**0.703**$^{+}$	**0.702**$^{+}$	$\underline{\mathbf{0.701}}^{+}$
	50	0.698	**0.697**	0.721^{-}	0.709^{-}	**0.695**	**0.694**	**0.695**	$\underline{\mathbf{0.693}}$
	100	0.69	0.692	0.716^{-}	0.705^{-}	**0.689**	**0.688**	$\underline{\mathbf{0.687}}$	**0.689**
	150	0.686	0.691^{-}	0.711^{-}	0.704^{-}	0.687	**0.686**	$\underline{\mathbf{0.685}}$	**0.686**
500	25	0.701	0.701	0.722^{-}	0.710^{-}	**0.697**$^{+}$	**0.698**$^{+}$	**0.698**$^{+}$	$\underline{\mathbf{0.697}}^{+}$
	50	0.692	0.695^{-}	0.716^{-}	0.705^{-}	**0.691**	$\underline{\mathbf{0.690}}$	**0.692**	**0.691**
	100	0.686	0.691^{-}	0.710^{-}	0.703^{-}	**0.685**	$\underline{\mathbf{0.685}}$	**0.686**	**0.686**
	150	0.684	0.689^{-}	0.707^{-}	0.702^{-}	**0.683**	$\underline{\mathbf{0.682}}$	**0.683**	**0.684**
1000	25	0.696	0.697	0.717^{-}	0.704^{-}	0.696	$\underline{\mathbf{0.694}}$	**0.696**	**0.696**
	50	$\underline{0.688}$	0.692^{-}	0.711^{-}	0.702^{-}	0.689^{-}	0.688	0.690	0.689
	100	$\underline{0.681}$	0.689^{-}	0.707^{-}	0.700^{-}	0.683^{-}	0.683^{-}	0.684^{-}	0.683^{-}
	150	$\underline{0.677}$	0.687^{-}	0.704^{-}	0.699^{-}	0.681^{-}	0.681^{-}	0.681^{-}	0.681^{-}

PP-AT has improved PP performance. Obviously, the training error of PP-AT are smaller than that of PP with all GP configurations. For TS-S, its training error is often not better than that of GP. This result is consistent with the result in Chapter 2 which illustrated that the statistics-based tournament selection techniques put less pressure on the improvement of training error in comparison to standard tournament selection.

The second metric used to analyse the performance of the tested GP systems is the prediction ability. For each run, the best solution is selected and evaluated on the testing data. The median of these values across 30 runs is shown in Table 3.10.

It can be seen from Table 3.10 that SAT-based methods increase the prediction ability of GP. The testing errors of SAT-GP, SAS-GP, SA

Table 3.10: Median of testing errors

Pop	Gen	GP	TS-S	PP	PP-AT	SAT-GP	SAS-GP	SA	DA
250	25	0.710	**0.710**	**0.702**	**0.707**	**0.691**	$\mathbf{0.686}^{+}$	$\underline{\mathbf{0.674}}^{+}$	**0.690**
	50	0.672	0.674	0.738^{-}	0.710^{-}	$\mathbf{0.662}^{+}$	$\underline{\mathbf{0.655}}^{+}$	$\mathbf{0.660}^{+}$	$\mathbf{0.663}^{+}$
	100	0.663	0.665	0.736^{-}	0.709^{-}	$\mathbf{0.654}^{+}$	$\underline{\mathbf{0.643}}^{+}$	$\mathbf{0.643}^{+}$	$\mathbf{0.655}^{+}$
	150	0.664	**0.659**	0.721^{-}	0.698^{-}	$\mathbf{0.651}^{+}$	$\mathbf{0.642}^{+}$	$\underline{\mathbf{0.640}}^{+}$	$\mathbf{0.644}^{+}$
500	25	0.696	**0.690**	0.705^{-}	0.706^{-}	**0.680**	**0.681**	$\underline{\mathbf{0.669}}^{+}$	$\mathbf{0.677}^{+}$
	50	0.674	0.675	0.738^{-}	0.707^{-}	$\underline{\mathbf{0.653}}$	$\mathbf{0.657}^{+}$	$\mathbf{0.661}^{+}$	**0.664**
	100	0.666	**0.663**	0.719^{-}	0.700^{-}	$\mathbf{0.644}^{+}$	$\underline{\mathbf{0.639}}^{+}$	$\mathbf{0.647}^{+}$	$\mathbf{0.646}^{+}$
	150	0.662	**0.657**	0.710^{-}	0.701^{-}	$\mathbf{0.645}^{+}$	$\underline{\mathbf{0.641}}^{+}$	$\mathbf{0.644}^{+}$	$\mathbf{0.644}^{+}$
1000	25	0.677	0.681	0.735^{-}	0.698^{-}	**0.672**	$\underline{\mathbf{0.653}}^{+}$	**0.670**	**0.670**
	50	0.665	0.669	0.725^{-}	0.687^{-}	**0.658**	$\underline{\mathbf{0.646}}^{+}$	$\mathbf{0.652}^{+}$	$\mathbf{0.652}^{+}$
	100	0.660	**0.659**	0.706^{-}	0.686^{-}	**0.650**	$\underline{\mathbf{0.634}}^{+}$	$\mathbf{0.638}^{+}$	$\mathbf{0.645}^{+}$
	150	0.664	**0.653**	0.708^{-}	0.686^{-}	$\mathbf{0.645}^{+}$	$\underline{\mathbf{0.631}}^{+}$	$\mathbf{0.634}^{+}$	$\mathbf{0.635}^{+}$

and DA are better than that of GP on all the experimental settings. Apparently, SAS-GP improved the performance greatly in comparison to other methods. The testing error of SAS-GP is the smallest on 8 GP configurations out of 12 GP configurations. For TS-S, it is also better than GP especially when large number of generations (100, 150) are used. More interestingly, although the training errors of TS-S, SAT-based methods are not better than that of GP at population size of 1000, the testing errors of them are much more convincing. Perhaps, the reason for the better performance on the testing data is due to their ability to obtain very simple solutions as shown in Table 3.11, and GP may be overfitted (the testing error of GP at generation 150 is bigger than that at generation 100, 0.664 vs 0.660).

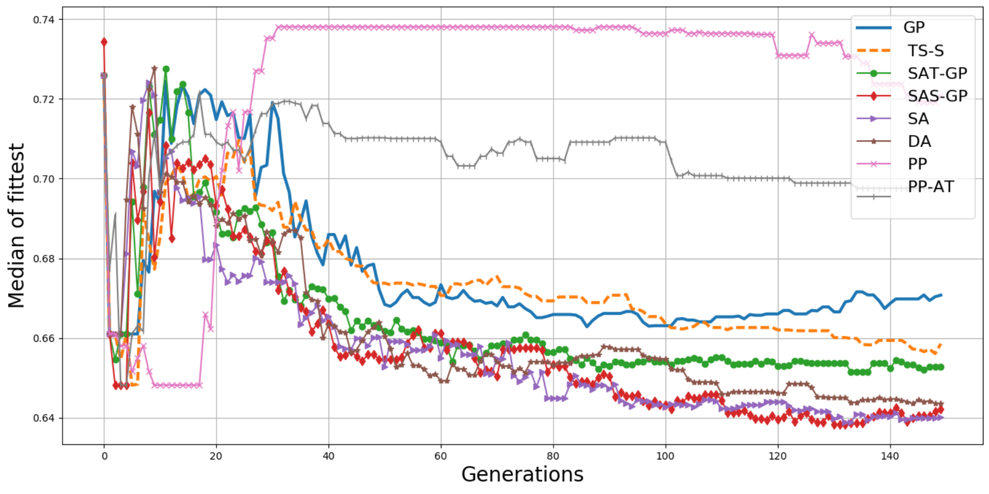
Conversely, the test error of both PP and PP-AT is worse than that of GP; even though, the combination of the semantic approximation technique with PP has increased the predicted performance of PP.

The results of Wilcoxon signed rank test with the confidence level of 95% again confirm the improvement of these methods. It is clear that the prediction ability of the SAT-based methods is significantly better than that of GP. Particularly, SAT-GP, SAS-GP, SA and DA are significantly better than GP on 6, 11, 11 and 9 GP configurations, respectively. On the other hand, GP is not significantly better than these SAT-based methods at any the GP parameter settings.

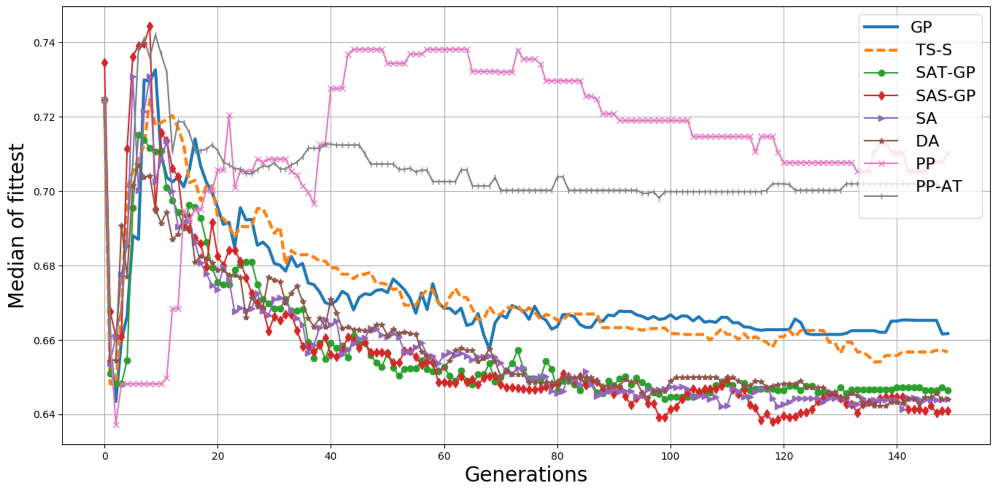
Figure 3.8 gives information about the testing error of the GP systems over the generations. This figure shows that the SAT-based methods quickly achieved a good testing error at the early generations and kept improving until the last generation. SAS-GP practically achieved the lowest testing error over the whole evolution process. The testing error produced from TS-S was approximately to that from GP in early generations, and continued improving for further generations while that from GP slightly increased at the last generations. The figure also confirms that PP-AT boosted the predictability of PP. The test error of PP-AT is much lower than that of PP during the evolution.

Next metric is the impact of the proposed methods on reducing the complexity of GP solutions and the GP code bloat. In each GP run, we record the size of the selected solutions. These values are then averaged over 30 runs and presented in Table 3.11.

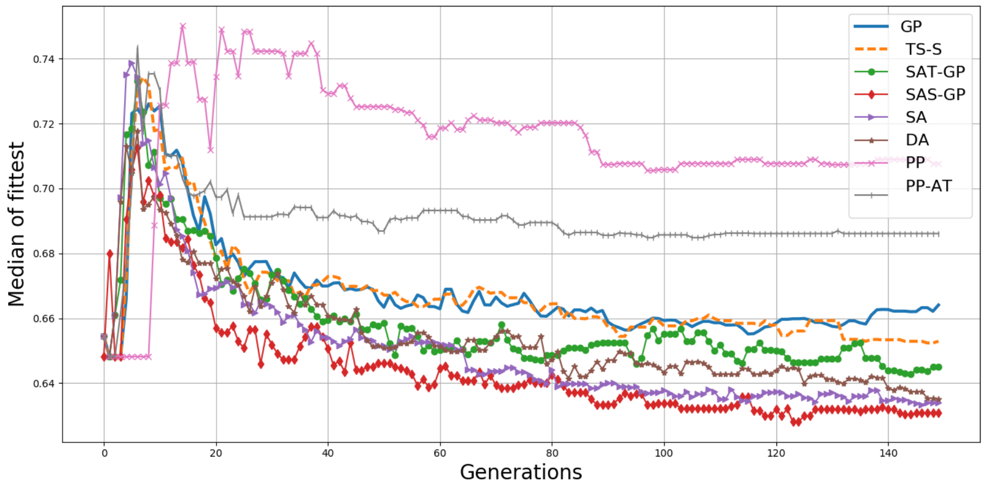
It can be seen from this table that the solutions found by all tested code bloat control methods are simpler than that of GP on most GP pa-



(a) At population size of 250



(b) At population size of 500



(c) At population size of 1000

Figure 3.8: Testing error over the generations.

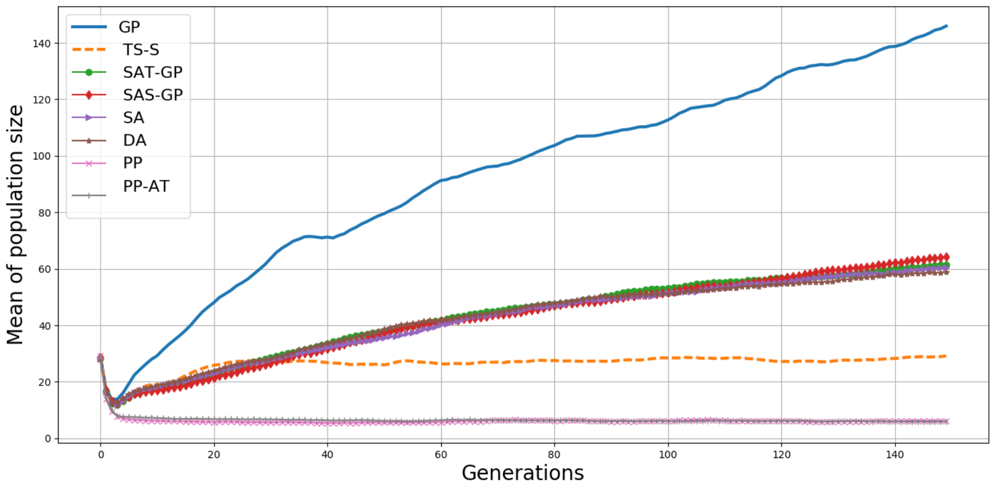
Table 3.11: Average of solution's size

Pop	Gen	GP	TS-S	PP	PP-AT	SAT-GP	SAS-GP	SA	DA
250	25	46.1	**35.9**	**9.7**+	**10.3** +	**26.4** +	**23.6** +	**25.1** +	**26.3** +
	50	78.9	**31.2** +	**10.7** +	**11.9** +	**39.0** +	**37.9** +	**36.9** +	**42.9** +
	100	111.7	**37.1** +	**11.7** +	**10.5** +	**53.3** +	**52.8** +	**54.8** +	**53.1** +
	150	141.6	**42.1** +	**14.4** +	**10.3** +	**66.5** +	**69.0** +	**62.3** +	**64.3** +
500	25	46.4	**29.2** +	**10.4** +	**10.9** +	**26.1** +	**25.2** +	**28.6** +	**27.7** +
	50	84.4	**35.4** +	**11.1** +	**10.7** +	**40.0** +	**37.8** +	**38.4** +	**40.8** +
	100	120.9	**32.9** +	**13.2** +	**12.0** +	**57.9** +	**52.6** +	**57.1** +	**56.6** +
	150	153.2	**38.7** +	**14.1** +	**12.7** +	**65.7** +	**64.0** +	**65.1** +	**67.6** +
1000	25	51.5	**34.5** +	**13.9** +	**13.9** +	**30.8** +	**27.6** +	**29.8** +	**28.6** +
	50	96.3	**36.6** +	**14.5** +	**15.4** +	**42.2** +	**40.7** +	**42.7** +	**37.9** +
	100	145.4	**45.0** +	**15.6** +	**15.7** +	**63.2** +	**54.5** +	**66.1** +	**53.6** +
	150	161.7	**52.5** +	**16.4** +	**17.5** +	**70.9** +	**63.3** +	**71.9** +	**65.2** +

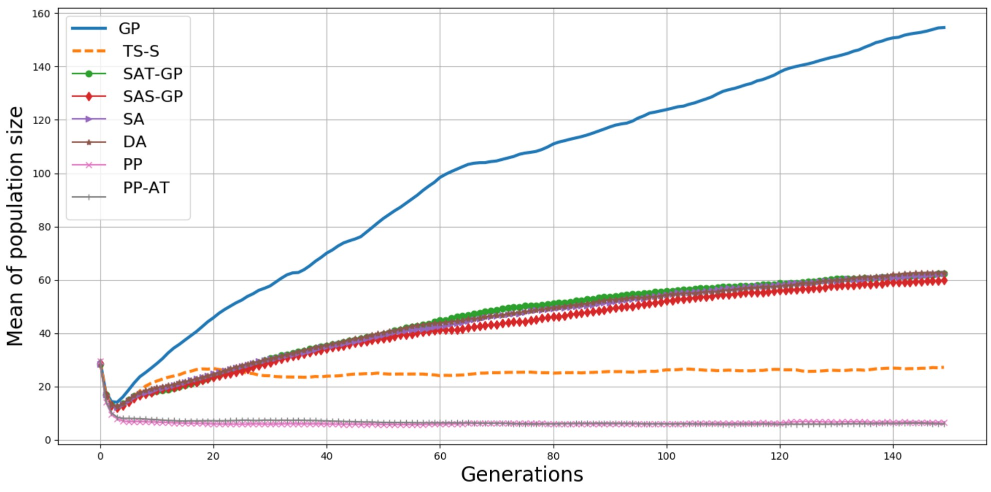
rameter settings. PP often obtains the simplest solutions. The solution size of PP-AT is roughly equal the that of PP. For SAT-based methods, the size of solutions created by them are also much smaller than that produced from GP.

The Wilcoxon test signed rank test also shows that the size of solutions obtained from these methods is significantly better than that of GP on most settings. Based on Occam's razor principle [75], obtaining a simple solution of these methods is probably one of the reasons for improving their predictive efficiency.

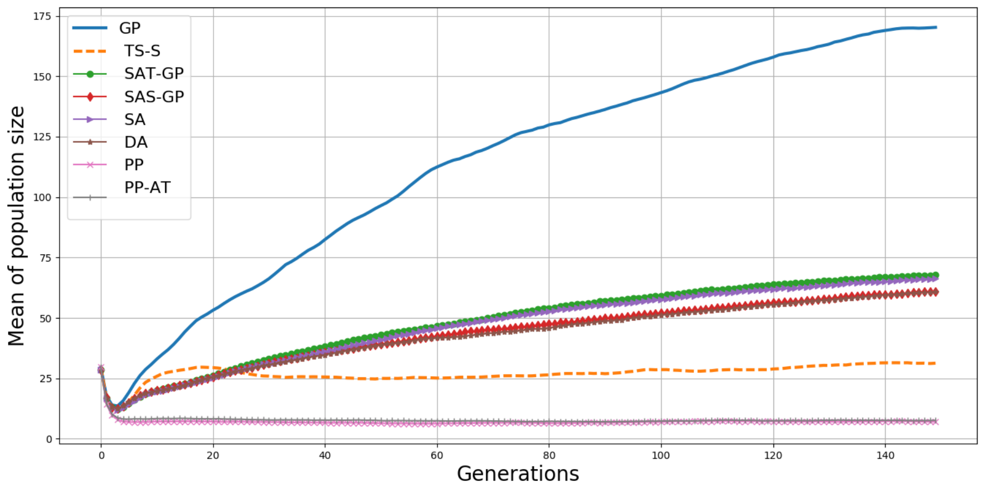
Figure 3.9 presents the average size of the population (the average size of the individuals in the population) over the evolutionary process. It can be observed from this figure that TS-S and SAT-based methods do not incur much code bloat phenomenon in GP population. The average



(a) At population size of 250



(b) At population size of 500



(c) At population size of 1000

Figure 3.9: Average size of population over the generations.

size of the population of TS-S is remained steady, and that of SAT-based methods are only slightly increased during the evolution. The graphs of PP and PP-AT equally overlap during evolution. Their average size of the population go down in the first generations and then remain stable until the last generations.

Conversely, the average size of the standard GP population quickly grows, and it is much larger than that of the others on all GP parameter settings. Overall, it is clear that all tested methods achieved their main objective for reducing code bloat phenomenon in GP system.

The last metric is the average running time of the tested GP systems. The total time needed to complete a GP run is recorded, and these values are then averaged over 30 runs. The results are showed in Table 3.12.

Table 3.12: Average running time in seconds

Pop	Gen	GP	TS-S	PP	PP-AT	SAT-GP	SAS-GP	SA	DA
250	25	5.3	7.0^{-}	5.5	10.0^{-}	**3.4**$^{+}$	**2.5**$^{+}$	9.4^{-}	**4.5**
	50	15.5	**15.1**	**3.2**$^{+}$	**5.2**$^{+}$	**9.6**$^{+}$	**7.1**$^{+}$	**8.8**$^{+}$	**10.8**$^{+}$
	100	42.3	**32.0**$^{+}$	**6.7**$^{+}$	**10.6**$^{+}$	**20.6**$^{+}$	**20.4**$^{+}$	**17.8**$^{+}$	**27.8**$^{+}$
	150	59.6	**36.1**$^{+}$	**11.5**$^{+}$	**18.3**$^{+}$	**29.4**$^{+}$	**33.2**$^{+}$	**31.1**$^{+}$	**44.3**$^{+}$
500	25	9.7	13.8^{-}	**4.4**$^{+}$	**6.4**$^{+}$	**6.5**$^{+}$	**5.8**$^{+}$	**7.6**$^{+}$	**11.6**$^{+}$
	50	32.8	**29.2**$^{+}$	**7.7**$^{+}$	**12.2**$^{+}$	**17.9**$^{+}$	**14.6**$^{+}$	**16.8**$^{+}$	**23.3**$^{+}$
	100	102.4	**59.8**$^{+}$	**16.3**$^{+}$	**22.9**$^{+}$	**34.4**$^{+}$	**39.3**$^{+}$	**37.1**$^{+}$	**56.9**$^{+}$
	150	138.4	**67.1**$^{+}$	**22.2**$^{+}$	**34.9**$^{+}$	**60.6**$^{+}$	**63.2**$^{+}$	**59.4**$^{+}$	**90.3**$^{+}$
1000	25	34.1	**24.6**$^{+}$	**10.9**$^{+}$	**14.6**$^{+}$	**13.5**$^{+}$	**12.2**$^{+}$	**15.8**$^{+}$	**21.2**$^{+}$
	50	95.2	**64.4**$^{+}$	**20.3**$^{+}$	**26.6**$^{+}$	**42.4**$^{+}$	**27.3**$^{+}$	**35.2**$^{+}$	**51.0**$^{+}$
	100	293.4	**131.1**$^{+}$	**36.5**$^{+}$	**52.1**$^{+}$	**99.3**$^{+}$	**70.6**$^{+}$	**75.0**$^{+}$	**124.8**$^{+}$
	150	355.0	**143.8**$^{+}$	**48.0**$^{+}$	**78.8**$^{+}$	**128.9**$^{+}$	**111.8**$^{+}$	**118.9**$^{+}$	**226.7**$^{+}$

It can be seen from this table that all tested bloat control methods, including TS-S, PP, PP-AT and SAT-based methods run faster than GP on most GP parameter settings. This is not surprising since the previous analysis has shown that these methods maintain a population which is much smaller in the average size in comparison to GP. Additionally, the average running time of PP is often the smallest. PP-AT also inherits this benefit; consequently, PP-AT is probably considered as the second fastest method.

In summary, the above analyses show that TS and SAT-based methods usually achieved the better performance in comparison to GP on four evaluative criteria on most the GP parameter settings. These evaluative criteria are predicting ability, lowering the complexity of the GP solutions, reducing code bloat phenomenon and running time. SAT-based methods also achieved better training error, especially at the population size of 250 and 500. Although PP-AT has not achieved good performance like TS-S and SAT-based methods, it has inherited the benefits and improved the performance of PP.

3.9. Conclusion

In this chapter, we proposed a new technique for generating a small tree in the form: $newTree = \theta \cdot sTree$ that is semantically similar to a target semantic vector. This technique is called *Semantic Approximation Technique* (SAT). Based on SAT, we proposed two approaches for lessening GP code bloat. The first method is *Subtree Approximation* (SA) in which a random subtree is chosen and replaced by a new tree of semantic approximation. The second method is *Desired Approximation*

(DA) where the new tree is grown to approximate the desired semantics of the selected subtree instead of its semantics.

Three configurations of SA and DA were tested on twenty-six symbolic regression problems. They were compared to standard GP, Prune and Plant [2] (PP), Statistics Tournament Selection with Size (TS-S) [C3], Random Desired Operator (RDO) [93] and four popular machine learning algorithms. The results showed that SA and DA outperform all tested GP models including GP, PP and RDO in improving the performance and the generalization of GP. Moreover, SA and DA found simpler solutions and imposed less overfitting and less code growth than the other GP methods. This property is very appealing since the previous bloat control methods in GP like PP [2] and neatGP [112] often did not improve the ability to fit the training data. Moreover, the performance of SA and DA is also competitive comparing to the best tested machine learning model (RF) on the selected datasets.

Besides, some other versions of SAT are introduced. Based on that, several other methods for reducing code bloat are proposed, including SAT-GP, SAS-GP and PP-AT. Then, all proposed bloat control methods based on semantics are applied in a real-world time series forecasting. The results illustrated that these methods help GP systems increase the performance on the time series forecasting problem.

CONCLUSIONS AND FUTURE WORK

The dissertation focuses on the selection stage in the evolution and the code bloat problem of GP. The overall goal was to improve GP performance by using semantic information. This goal was successfully achieved by developing a number of new methods based on incorporating semantics into GP evolutionary process. The proposed methods were evaluated and compared with existing methods on a large set of regression problems and a real-world time series forecasting. Results show that the proposed methods are able to promote semantic diversity in GP population, improve GP performance and address GP code bloat problem. This section gives a summary of the main contributions of the dissertation, then presents some limitations and possible future extensions derived from the dissertation.

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

- Three semantic tournament selection are proposed, including TS-R, TS-S and TS-P. The methods are based on a new comparison proposal between individuals using a statistical analysis. A statistical hypothesis test employs information from the individual's error vector to test the differences among individuals in GP. Additionally, for further improvement, TS-S is combined with the recently proposed semantic crossover, RDO, and the resulting method is called TS-RDO. These methods are tested on twenty-six regression problems and their noisy variants. The experimental results demonstrate the benefit of the proposed methods in promoting semantic diversity, reducing GP code growth and improving the generalisation behaviour of GP solutions when compared to standard tournament selection, a similar selection technique and a state of the art bloat control approach.

- A novel semantic approximation technique, SAT is proposed that allows to grow a small tree in the form $newTree = \theta \cdot sTree$ ($sTree$ is a small randomly generated tree) with the semantics approximate to a given target semantics. Besides, two other versions of SAT are also introduced wherein $sTree$ is a random terminal taken from the terminal set, or $sTree$ is a small tree taken from the pre-defined library.
- Two methods based on semantic approximation technique for reducing GP code bloat are proposed. The first method called SA replaces a random subtree in an individual by a smaller tree of approximate semantics. The second method called DA replaces a random subtree by a smaller tree that is semantically approximate to the desired semantics. Moreover, three other bloat control methods based on the variants of SAT, including SAT-GP, SAS-GP and PP-AT are intro-

duced. The performance of the bloat control strategies is examined on a large set of regression problems and a real-world time series forecasting. The experimental results showed that the proposed methods improve the performance of GP and specifically reduce code bloat compared to standard GP and several recent bloat control methods in GP. Furthermore, the performance of the proposed approaches is competitive with the best machine learning technique among the four tested machine learning algorithms.

In addition to a new variant of GP structure is proposed in the process of carrying out the dissertation. A number of subdatasets are sampled from the training data and a subpopulation is evolved on each of these datasets for a pre-defined generation. The subpopulations are then combined to form a full population that is evolved on the full training dataset for the rest generations.

However, the dissertation is subject to some limitations. First, the proposed methods are based on the concepts of sampling semantics that is only defined for the problems in which the input and output are continuous real-valued vectors. Subsequently, these methods were only applied to the real-valued symbolic regression problems and leaving other domains like reinforcement learning problems and classification problems an open question. Second, the semantic selection methods use the statistical analysis of GP error vectors. In the experiments, we use Wilcoxon Signed Rank Test to analyse the error vectors. Nevertheless, selecting an appropriate statistical test will increase the performance of the proposed methods. The dissertation lacks examining the distribution of GP error vectors. Therefore, in the future, we will examine this. Third, two approaches for reducing GP code bloat, SA and DA add two more parameters (max depth of $sTree$ and the portion of GP population for pruning) to GP systems. Currently, these parameters were experimentally determined and they might not be the best choices for these problems and for others.

Building upon this research, there are a number of directions for future work arisen from the dissertation. Firstly, we will conduct research to reduce the above limitations of the dissertation. Secondly, statistical analysis was used only to enhance selection. It is also possible that statistical analysis can be employed in other phases of the GP algorithm, for example, in model selection [129]. Thirdly, SAT was used for lessening code bloat in GP. Nevertheless, this technique can also be used for designing new genetic operators be similar to RDO [93]. Finally, in terms of applications, all proposed methods in the dissertation can be applied to any problem domain where the output is a single real-valued number. In this dissertation, we focused exclusively on GP's most popular problem domain, symbolic regression, in the future we will extend them to a wider range of real-world applications including classification and problems of bigger datasets to better understand their weakness and strength.

PUBLICATIONS

[C1] **Chu, T.H.**, Nguyen, Q.U., ONeill, M.: *Tournament selection based on statistical test in genetic programming.* In: The proceeding of International Conference on Parallel Problem Solving from Nature. pp. 303–312. Springer (2016).

[C2] **Chu, T.H.**, Nguyen, Q.U.: *Reducing code bloat in genetic programming based on subtree substituting technique.* In: The proceeding of 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES). pp. 25–30. IEEE (2017).

[C3] **Chu, T.H.**, Nguyen, Q.U., O'Neill, M.: *Semantic tournament selection for genetic programming based on statistical analysis of error vectors.* Information Sciences (ISI-SCI, Q1, IF=5.524) 436, 352–366 (2018).

[C4] **Chu, T.H.**, Nguyen, Q.U.: *Sampling method for evolving multiple subpopulations in genetic programming.* Journal of Science and Technology: The Section on Information and Communication Technology 12, 5–16 (2018).

[C5] **Chu, T.H.**, Nguyen, Q.U., Cao, V.L.: *Semantics based substituting technique for reducing code bloat in genetic programming.* In: Proceedings of the Ninth International Symposium on Information and Communication Technology. pp. 77 – 83. ACM (2018).

[C6] **Chu, T.H.**: *Semantic approximation based operator for reducing* *code bloat in genetic programming.* In: The 14th Young Researchers Conference. pp. 3–4. and Under review in Journal of Science and Technology: The Section on Information and Communication Technology, Le Quy Don Technical University (2019).

[C7] Nguyen, Q.U, **Chu, T.H.**: *Semantic Approximation for Reducing Code Bloat in Genetic Programming.* Under review in: Swarm and Evolutionary Computation(ISI-SCIE, Q1, IF=6.330)(2019).

BIBLIOGRAPHY

[1] Al-Betar, M.A., Awadallah, M.A., Faris, H., Aljarah, I., Hammouri, A.I.: Natural selection methods for grey wolf optimizer. Expert Systems with Applications 113, 481–498 (2018)

[2] Alfaro-Cid, E., Esparcia-Alcázar, A., Sharman, K., de Vega, F.F., Merelo, J.: Prune and plant: a new bloat control method for genetic programming. In: Hybrid Intelligent Systems 2008. pp. 31–35. IEEE (2008)

[3] Alfaro-Cid, E., Merelo, J.J., de Vega, F.F., Esparcia-Alcázar, A.I., Sharman, K.: Bloat control operators and diversity in genetic programming: A comparative study. Evolutionary Computation 18(2), 305–332 (2010)

[4] Bache, K., Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml

[5] Bäck, T.: Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In: Proceedings of the first IEEE conference on evolutionary computation. IEEE World Congress on Computational Intelligence. pp. 57–62. IEEE (1994)

[6] Beadle, L., Johnson, C.G.: Semantically driven crossover in genetic programming. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence). pp. 111–116. IEEE (2008)

[7] Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10(3), 307–337 (2009)

[8] Beadle, L., Johnson, C.G.: Semantically driven mutation in genetic programming. In: 2009 IEEE Congress on Evolutionary Computation. pp. 1336–1342. IEEE (2009)

[9] Belpaeme, T.: Evolution of visual feature detectors. In: University of Birmingham School of Computer Science technical. Citeseer (1999)

[10] Blickle, T., Thiele, L.: A comparison of selection schemes used in evolutionary algorithms. Evolutionary Computation 4(4), 361–394 (1996)

[11] Castelli, M., Castaldi, D., Giordani, I., Silva, S., Vanneschi, L., Archetti, F., Maccagnola, D.: An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics. In: Portuguese Conference on Artificial Intelligence. pp. 78–89. Springer (2013)

[12] Castelli, M., Manzoni, L., Silva, S., Vanneschi, L., Popovic, A.: The influence of population size in geometric semantic gp. Swarm and Evolutionary Computation 32, 110–120 (2017)

[13] Cavaretta, M.J., Chellapilla, K.: Data mining using genetic programming: the implications of parsimony on generalization error. In: Proceedings of the 1999 Congress on Evolutionary Computation. vol. 2, p. 1337 Vol. 2 (1999)

[14] Chen, Q., Xue, B., Mei, Y., Zhang, M.: Geometric semantic crossover with an angle-aware mating scheme in genetic programming for symbolic regression. In: European Conference on Genetic Programming. pp. 229–245. Springer (2017)

[15] Chen, Q., Xue, B., Zhang, M.: Improving generalisation of genetic programming for symbolic regression with angle-driven geometric semantic operators. IEEE Transactions on Evolutionary Computation (2018)

[16] Chen, Q., Zhang, M., Xue, B.: Geometric semantic genetic programming with perpendicular crossover and random segment mutation for symbolic regression. In: Asia-Pacific Conference on Simulated Evolution and Learning. pp. 422–434. Springer (2017)

[17] Cumming, G.: Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis. Routledge (2012)

[18] Data, K.: Corporación favorita grocery sales forecasting (2018), https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data

[19] Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1(1), 3–18 (2011)

[20] Dick, G., Whigham, P.A.: Controlling bloat through parsimonious elitist replacement and spatial structure. In: European Conference on Genetic Programming. pp. 13–24. Springer (2013)

[21] Dignum, S., Poli, R.: Operator equalisation and bloat free gp. Lecture Notes in Computer Science 4971, 110–121 (2008)

[22] Dijkstra, E.W., Scholten, C.S.: Predicate calculus and program semantics. Springer Science & Business Media (2012)

[23] Dou, T., Rockett, P.: Semantic-based local search in multiobjective genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 225–226. ACM (2017)

[24] Eiben, A.E., Smith, J.E., et al.: Introduction to evolutionary computing, vol. 53. Springer (2003)

[25] Euzenat, J., Shvaiko, P., et al.: Ontology matching, vol. 18. Springer (2007)

[26] Fang, Y., Li, J.: A review of tournament selection in genetic programming. In: International Symposium on Intelligence Computation and Applications. pp. 181–192. Springer (2010)

[27] Forstenlechner, S., Nicolau, M., Fagan, D., O'Neill, M.: Introducing semantic-clustering selection in grammatical evolution. In: Proceedings of the Companion

Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1277–1284. ACM (2015)

[28] Fracasso, J.V.C., Von Zuben, F.J.: Multi-objective semantic mutation for genetic programming. In: 2018 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2018)

[29] Galvan-Lopez, E., Cody-Kenny, B., Trujillo, L., Kattan, A.: Using semantics in the selection mechanism in genetic programming: a simple method for promoting semantic diversity. In: 2013 IEEE Congress on Evolutionary Computation. pp. 2972–2979. IEEE (2013)

[30] Gandomi, A.H., Alavi, A.H., Ryan, C.: Handbook of genetic programming applications. Springer (2015)

[31] Gardner, M.A., Gagné, C., Parizeau, M.: Controlling code growth by dynamically shaping the genotype size distribution. Genetic Programming and Evolvable Machines 16(4), 455–498 (2015)

[32] Gathercole, C.: An investigation of supervised learning in genetic programming. Ph.D. thesis (1998)

[33] Ghodrat, M.A., Givargis, T., Nicolau, A.: Equivalence checking of arithmetic expressions using fast evaluation. In: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems. pp. 147–156. ACM (2005)

[34] Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms 1, 69–93 (1991)

[35] Hara, A., Kushida, J.i., Tanemura, R., Takahama, T.: Deterministic crossover based on target semantics in geometric semantic genetic programming. In: 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI). pp. 197–202. IEEE (2016)

[36] Harper, R.: Practical foundations for programming languages. Cambridge University Press (2016)

[37] Helmuth, T., McPhee, N.F., Spector, L.: Effects of lexicase and tournament selection on diversity recovery and maintenance. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. pp. 983–990. ACM (2016)

[38] Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. IEEE Transactions on Evolutionary Computation 19(5), 630–643 (2015)

[39] Hingee, K., Hutter, M.: Equivalence of probabilistic tournament and polynomial ranking selection. In: 2008 IEEE Congress on Evolutionary Computation. pp. 564–571. IEEE (2008)

[40] Iba, H.: Evolutionary approach to deep learning. In: Evolutionary Approach to Machine Learning and Deep Neural Networks, pp. 77–104. Springer (2018)

[41] Juárez-Smith, P., Trujillo, L.: Integrating local search within neat-gp. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. pp. 993–996. ACM (2016)

[42] Julstrom, B.A., Robinson, D.H.: Simulating exponential normalization with weighted k-tournaments. In: Proceedings of the 2000 Congress on Evolutionary Computation. vol. 1, pp. 227–231. IEEE (2000)

[43] Kanji, G.K.: 100 Statistical Tests. SAGE Publications (1999)

[44] Kattan, A., Agapitos, A., Ong, Y.S., Alghamedi, A.A., O'Neill, M.: Gp made faster with semantic surrogate modelling. Information Sciences 355-356, 169–185 (2016)

[45] Kattan, A., Ong, Y.S.: Surrogate genetic programming: A semantic aware evolutionary search. Information Sciences 296, 345–359 (2015)

[46] Kelly, S., Heywood, M.I.: Emergent solutions to high-dimensional multitask reinforcement learning. Evolutionary computation 26(3), 347–380 (2018)

[47] Kelly, S., Smith, R.J., Heywood, M.I.: Emergent policy discovery for visual reinforcement learning through tangled program graphs: A tutorial. Genetic programming theory and practice XVI pp. 37–57 (2019)

[48] Kim, J.J., Zhang, B.T.: Effects of selection schemes in genetic programming for time series prediction. In: Proceedings of the Congress on Evolutionary Computation. vol. 1, pp. 252–258 (1999)

[49] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. The MIT Press (1992)

[50] Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and Computing 4(2), 87–112 (1994)

[51] Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 987–994. ACM (2009)

[52] Krawiec, K., Pawlak, T.: Locally geometric semantic crossover. In: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation. pp. 1487–1488. ACM (2012)

[53] Krawiec, K., Pawlak, T.: Approximating geometric crossover by semantic backpropagation. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 941–948. ACM (2013)

[54] Krawiec, K., Pawlak, T.: Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. Genetic Programming and Evolvable Machines 14(1), 31–63 (2013)

[55] La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. Evolutionary computation pp. 1–26 (2018)

[56] La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 741–748. ACM (2016)

[57] Le, T.A., Chu, T.H., Nguyen, Q.U., Nguyen, X.H.: Malware detection using genetic programming. In: the 2014 Seventh IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA). pp. 1–6. IEEE (2014)

[58] Luke, S., Panait, L.: Fighting bloat with nonparametric parsimony pressure. Parallel Problem Solving from Nature VII pp. 411–421 (2002)

[59] Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. pp. 829–836. Morgan Kaufmann Publishers Inc. (2002)

[60] Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3), 309–344 (2006)

[61] Maghsoodlou, S., Noroozi, B., Haghi, A.: Application of genetic programming approach for optimization of electrospinning parameters. Polymers Research Journal 11(1), 17–25 (2017)

[62] Mariot, L., Picek, S., Leporati, A., Jakobovic, D.: Cellular automata based s-boxes. Cryptography and Communications 11(1), 41–62 (2019)

[63] Martin, P., Poli, R.: Crossover operators for a hardware implementation of gp using fpgas and handel-c. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. pp. 845–852. Morgan Kaufmann Publishers Inc. (2002)

[64] Martins, J.F., Oliveira, L.O.V., Miranda, L.F., Casadei, F., Pappa, G.L.: Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1151–1158. ACM (2018)

[65] McConaghy, T.: Ffx: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, chap. 13, pp. 235–260. Springer (2011)

[66] Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'neill, M.: Grammar-based genetic programming: a survey. Genetic Programming and Evolvable Machines 11(3-4), 365–396 (2010)

[67] McPhee, N., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: Proceedings of 11th European Conference on Genetic Programming. pp. 134–145. Springer (2008)

[68] Metevier, B., Saini, A.K., Spector, L.: Lexicase selection beyond genetic programming. In: Genetic Programming Theory and Practice XVI, pp. 123–136. Springer (2019)

[69] Miller, J.F.: Cartesian genetic programming. In: Cartesian Genetic Programming, pp. 17–34. Springer (2011)

[70] Miller, J.F., Thomson, P.: Cartesian genetic programming. In: European Conference on Genetic Programming. pp. 121–132. Springer (2000)

[71] Mitchell, T.M.: Machine Learning. McGraw-Hill Science, New York (1997)

[72] Moraglio, A.: An efficient implementation of gsgp using higher-order functions and memoization. Semantic Methods in Genetic Programming, Ljubljana, Slovenia 13 (2014)

[73] Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 21–31. Springer (2012)

[74] Naredo, E., Trujillo, L., Legrand, P., Silva, S., Munoz, L.: Evolving genetic programming classifiers with novelty search. Information Sciences 369, 347–367 (2016)

[75] Needham, S., Dowe, D.L.: Message length as an effective ockham's razor in decision tree induction. In: International Workshop on Artificial Intelligence and Statistics (AISTATS). Society for Artificial Intelligence and Statistics (2001)

[76] Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic aware crossover for genetic programming: the case for real-valued function regression. In: European Conference on Genetic Programming. pp. 292–302. Springer (2009)

[77] Nguyen, Q.U., Nguyen, X.H., O'Neill, M., McKay, B.: Semantics based crossover for boolean problems. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 869–876. ACM (2010)

[78] Nguyen, Q.U., Nguyen, X.H., O'Neill, M., McKay, R.I., Dao, N.P.: On the roles of semantic locality of crossover in genetic programming. Information Sciences 235, 195–213 (2013)

[79] Nguyen, Q.U., Nguyen, X.H., O'Neill, M., McKay, R.I., Galvan-Lopez, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12(2), 91–119 (2011)

[80] Nguyen, Q.U., O'Neill, M., Nguyen, X.H.: Predicting the tide with genetic programming and semantic-based crossovers. In: 2010 Second International Conference on Knowledge and Systems Engineering. pp. 89–95. IEEE (2010)

[81] Nguyen, Q.U., O'Neill, M., Nguyen, X.H.: Examining semantic diversity and semantic locality of operators in genetic programming. Ph.D. thesis, University College Dublin (2011)

[82] Nguyen, Q.U., Pham, T.A., Nguyen, X.H., McDermott, J.: Subtree semantic geometric crossover for genetic programming. Genetic Programming and Evolvable Machines 17(1), 25–53 (2016)

[83] Oksanen, K., Hu, T.: Lexicase selection promotes effective search and behavioural diversity of solutions in linear genetic programming. In: 2017 IEEE Congress on Evolutionary Computation (CEC). pp. 169–176. IEEE (2017)

[84] Oliveira, L.O.V., Casadei, F., Pappa, G.L.: Strategies for improving the distribution of random function outputs in gsgp. In: European Conference on Genetic Programming. pp. 164–177. Springer (2017)

[85] Oliveira, L.O.V., Miranda, L.F., Pappa, G.L., Otero, F.E., Takahashi, R.H.: Reducing dimensionality to improve search in semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 375–385. Springer (2016)

[86] Oliveira, L.O.V., Otero, F.E., Pappa, G.L.: A dispersion operator for geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 773–780. ACM (2016)

[87] Oltean, M., Groşan, C., Dioşan, L., Mihăilă, C.: Genetic programming with linear representation: a survey. International Journal on Artificial Intelligence Tools 18(02), 197–238 (2009)

[88] O'Neill, M., Vanneschi, L., Gustafson, S.M., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11, 339–363 (2010)

[89] Panait, L., Luke, S.: Alternative bloat control methods. In: Genetic and Evolutionary Computation Conference. pp. 630–641. Springer (2004)

[90] Pawlak, T.P., Krawiec, K.: Progress properties and fitness bounds for geometric semantic search operators. Genetic Programming and Evolvable Machines 17(1), 5–23 (2016)

[91] Pawlak, T.P., Krawiec, K.: Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. Evolutionary computation 26(2), 177–212 (2018)

[92] Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and comparative analysis of geometric semantic crossovers. Genetic Programming and Evolvable Machines 16(3), 351–386 (2015)

[93] Pawlak, T.P., Wieloch, B., Krawiec, K.: Semantic backpropagation for designing search operators in genetic programming. IEEE Transactions on Evolutionary Computation 19(3), 326–340 (2015)

[94] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Sklearn tutorial [online] (2011), https://scikit-learn.org/stable/ Accessed: 2019-11-24

[95] Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. Genetic programming pp. 43–76 (2003)

[96] Poli, R.: Covariant tarpeian method for bloat control in genetic programming. Genetic Programming Theory and Practice VIII pp. 71–89 (2011)

[97] Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming. Lulu. com (2008)

[98] Poli, R., McPhee, N.F., Citi, L., Crane, E.: Memory with memory in tree-based genetic programming. In: European Conference on Genetic Programming. pp. 25–36. Springer (2009)

[99] Purohit, A., Choudhari, N.S., Tiwari, A.: Code bloat problem in genetic programming. International Journal of Scientific and Research Publications 3(4), 1612 (2013)

[100] Rumpf, D.L.: Statistics for dummies. Technometrics 46(3) (2004)

[101] Sáez, J.A., Galar, M., Luengo, J., Herrera, F.: Tackling the problem of classification with noisy data using multiple classifier systems: Analysis of the performance and robustness. Information Sciences 247, 1–20 (2013)

[102] Sáez, J.A., Galar, M., Luengo, J., Herrera, F.: Analyzing the presence of noise in multi-class problems: alleviating its influence with the one-vs-one decomposition. Knowledge and Information Systems 38(1), 179–206 (2014)

[103] Sara, S., Leonardo, V.: The importance of being flat-studying the program length distributions of operator equalisation. Genetic Programming Theory and Practice IX pp. 211–233 (2011)

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

[105] Silva, S., Dignum, S.: Extending operator equalisation: Fitness based self adaptive length distribution for bloat free gp. In: European Conference on Genetic Programming. pp. 159–170. Springer (2009)

[106] Silva, S., Dignum, S., Vanneschi, L.: Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genetic Programming and Evolvable Machines 13(2), 197–238 (2012)

[107] Silva, S., Vanneschi, L.: Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 1115–1122. ACM (2009)

[108] Sokolov, A., Whitley, D.: Unbiased tournament selection. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation. pp. 1131–1138. ACM (2005)

[109] Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 497–504. ACM (2017)

[110] Szubert, M., Kodali, A., Ganguly, S., Das, K., Bongard, J.C.: Semantic forward propagation for symbolic regression. In: International Conference on Parallel Problem Solving from Nature. pp. 364–374. Springer (2016)

[111] Trujillo, L., Emigdio, Z., Juárez-Smith, P.S., Legrand, P., Silva, S., Castelli, M., Vanneschi, L., Schütze, O., Muñoz, L., et al.: Local search is underused in genetic programming. Genetic Programming Theory and Practice XIV pp. 119–137 (2018)

[112] Trujillo, L., Muñoz, L., Galván-López, E., Silva, S.: neat genetic programming: Controlling bloat naturally. Information Sciences 333, 21–43 (2016)

[113] Trujillo, L., Olague, G., Lutton, E., de Vega, F.F., Dozal, L., Clemente, E.: Speciation in behavioral space for evolutionary robotics. Journal of Intelligent and Robotic Systems 64(3-4), 323–351 (2011)

[114] Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A new implementation of geometric semantic gp and its application to problems in pharmacokinetics. In: European Conference on Genetic Programming. pp. 205–216. Springer (2013)

[115] Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 877–884. ACM (2010)

[116] Vanneschi, L., Castelli, M., Silva, S.: A survey of semantic methods in genetic programming. Genetic Programming and Evolvable Machines 15(2), 195–214 (2014)

[117] Vanneschi, L., Galvao, B.: A parallel and distributed semantic genetic programming system. In: 2017 IEEE Congress on Evolutionary Computation (CEC). pp. 121–128. IEEE (2017)

[118] Vyas, R., Bapat, S., Goel, P., Karthikeyan, M., Tambe, S.S., Kulkarni, B.D.: Application of genetic programming gp formalism for building disease predictive models from protein-protein interactions ppi data. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 15(1), 27–37 (2018)

[119] Whigham, P.A., Dick, G.: Implicitly controlling bloat in genetic programming. IEEE Transaction on Evolutionary Computation 14(2), 173–190 (2010)

[120] White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaskowski, W., O'Reilly, U.M., Luke, S.: Better GP benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines 14(1), 3–29 (2013)

[121] Wilson, D.G., Cussat-Blanc, S., Luga, H., Miller, J.F.: Evolving simple programs for playing atari games. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 229–236. ACM (2018)

[122] Xie, H.: Diversity control in gp with adf for regression tasks. In: Australasian Joint Conference on Artificial Intelligence. pp. 1253–1257. Springer (2005)

[123] Xie, H., Zhang, M.: Impacts of sampling strategies in tournament selection for genetic programming. Soft Computing 16(4), 615–633 (2012)

[124] Xie, H., Zhang, M.: Parent selection pressure auto-tuning for tournament selection in genetic programming. IEEE Transactions on Evolutionary Computation 17(1), 1–19 (2013)

[125] Xie, H., Zhang, M., Andreae, P.: Automatic selection pressure control in genetic programming. In: Sixth International Conference on Intelligent Systems Design and Applications. vol. 1, pp. 435–440. IEEE (2006)

[126] Xie, H., Zhang, M., Andreae, P., Johnson, M.: An analysis of multi-sampled issue and no-replacement tournament selection. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation. pp. 1323–1330. ACM (2008)

[127] Xie, H., Zhang, M., Andreae, P., Johnston, M.: Is the not-sampled issue in tournament selection critical? In: 2008 IEEE Congress on Evolutionary Computation. pp. 3710–3717. IEEE (2008)

[128] Yoo, S., Xie, X., Kuo, F.C., Chen, T.Y., Harman, M.: Human competitiveness of genetic programming in spectrum-based fault localisation: theoretical and empirical analysis. ACM Transactions on Software Engineering and Methodology (TOSEM) 26(1), 4 (2017)

[129] Žegklitz, J., Pošík, P.: Model selection and overfitting in genetic programming: Empirical study. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1527–1528. ACM (2015)

Appendix
Remaining results of the statistics tournament selection methods

This appendix presents the remaining results of the methods tested in Chapter 2. The table results include:

- Mean best fitness on training noise data with tour size=3 and tour size=7.
- Average of solutions size on training noise data with tour size=3 and tour size=7.
- Mean of best fitness of GP and three semantics tournament selections with tour size=5.
- Median of testing error of GP and three semantics tournament selections with tour size=5.
- Average of solution's size of GP and three semantics tournament selections with tour size=5.
- Mean of best fitness of TS-RDO and four other techniques with tour size=5.
- Median of fittest of TS-RDO and four other techniques with tour size=5.
- Average of solutions size of TS-RDO and four other techniques with tour size=5.

Table A.1: Mean best fitness on training noise data with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RDO
A. **Benchmarking Problems**										
F1	2.06	4.78$^-$	3.41$^-$	**0.15**$^+$	2.43	1.69	4.78$^-$	3.55$^-$	**0.19**$^+$	3.38$^-$
F2	0.22	0.41$^-$	0.57$^-$	**0.05**$^+$	**0.21**	0.22	0.41$^-$	0.58$^-$	**0.06**$^+$	0.39$^-$
F3	5.39	13.11$^-$	6.63	**0.17**$^+$	**0.91**$^+$	4.75	13.11$^-$	6.33	**0.21**$^+$	**1.52**$^+$
F4	0.10	0.17$^-$	0.11$^-$	**0.08**$^+$	**0.09**$^+$	0.10	0.17$^-$	0.12$^-$	**0.08**$^+$	**0.10**
F5	0.14	0.16$^-$	0.14	**0.13**	0.15$^-$	0.14	0.16$^-$	0.14	0.14	0.15$^-$
F6	0.76	1.00$^-$	1.23$^-$	**0.28**$^+$	**0.53**$^+$	0.62	1.00$^-$	1.26$^-$	**0.27**$^+$	**0.61**
F7	0.48	0.54$^-$	0.56$^-$	**0.26**$^+$	**0.45**	0.45	0.54$^-$	0.57$^-$	**0.27**$^+$	0.46
F8	66.8	69.2$^-$	67.2$^-$	**65.9**	67.3 $^-$	66.5	69.2$^-$	67.3$^-$	**66.0**	67.4$^-$
F9	3.99	5.64$^-$	4.61$^-$	**2.95**$^+$	**3.22**	5.40	5.64$^-$	6.74$^-$	**2.96**$^+$	**3.34**
F10	9.93	10.9	**6.82**	**2.72**$^+$	**2.85**$^+$	7.96	10.9$^-$	**6.98**	**3.58**$^+$	**2.71**$^+$
F11	0.21	0.30$^-$	0.21	**0.18**$^+$	**0.19**$^+$	0.22	0.30$^-$	**0.21**$^+$	**0.18**	**0.19**$^+$
F12	7.15	7.52$^-$	7.17$^-$	**6.76**$^+$	**6.98**	7.03	7.52$^-$	7.17$^-$	**6.81**$^+$	7.06$^-$
F13	0.88	0.93$^-$	0.89$^-$	**0.87**	0.89$^-$	0.89	0.93$^-$	0.89$^-$	**0.87**	**0.89**$^+$
F14	102.6	109.4$^-$	104.5$^-$	**94.9**$^+$	**102.4**	103.1	109.4$^-$	**102.7**$^+$	**96.2**$^+$	103.6$^-$
F15	3.04	3.95$^-$	**3.02**	**1.86**$^+$	**2.01**$^+$	2.52	3.95$^-$	2.65$^-$	**1.86**$^+$	**2.02**
B. **UCI Problems**										
F16	19.3	23.82$^-$	20.0	**9.49**$^+$	**9.72**$^+$	18.6	23.8$^-$	19.6	**9.37**$^+$	**9.78**$^+$
F17	3.97	4.31$^-$	4.36$^-$	**2.82**$^+$	**3.69**	3.62	4.31$^-$	4.37$^-$	**2.57**$^+$	3.78
F18	45.8	56.6$^-$	45.8	**34.6**$^+$	**35.6**$^+$	45.4	56.6$^-$	45.7	**33.9**$^+$	**35.7**$^+$
F19	26.0	28.50$^-$	31.5$^-$	**22.1**$^+$	28.3$^-$	24.3	28.5$^-$	31.7$^-$	**22.2**	28.6$^-$
F20	16.6	16.9$^-$	16.7$^-$	**15.0**$^+$	**15.6**$^+$	16.3	16.9$^-$	16.7$^-$	**14.8**$^+$	**15.7**$^+$
F21	4.49	4.68$^-$	4.54	**4.05**$^+$	**4.18**$^+$	4.41	4.68$^-$	4.51	**4.00**$^+$	**4.19**$^+$
F22	3.44	4.22$^-$	3.75$^-$	**2.78**$^+$	3.45	3.19	4.22$^-$	3.85$^-$	**2.80**$^+$	3.57$^-$
F23	5.07	7.14$^-$	**5.07**	**1.59**$^+$	**3.03**$^+$	4.09	7.14$^-$	8.81$^-$	**1.36**$^+$	**3.68**
F24	11.6	13.6$^-$	14.3$^-$	**5.50**$^+$	**11.0**	10.1	13.6$^-$	15.6$^-$	**4.57**$^+$	11.8$^-$
F25	5.46	6.79$^-$	7.04$^-$	**2.33**$^+$	**4.77**	4.81	6.79$^-$	7.48$^-$	**2.07**$^+$	5.49$^-$
F26	53.12	53.64$^-$	53.25	**52.63**	**53.07**	53.23	53.64$^-$	53.52$^-$	**52.85**	53.31

Table A.2: Average of solutions size on training noise data with tour-size=3 (the left) and tour-size=7 (the right)

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S
A. **Benchmarking Problems**								
F1	273	**123** $^+$	**120** $^+$	**248**	$\underline{\mathbf{92}}^+$	295	**123** $^+$	**100** $^+$
F2	184	**65** $^+$	$\underline{\mathbf{35}}^+$	**174**	**97** $^+$	168	**65** $^+$	$\underline{\mathbf{38}}^+$
F3	260	**103** $^+$	**128** $^+$	**190** $^+$	$\underline{\mathbf{98}}^+$	260	**103** $^+$	**104** $^+$
F4	250	$\underline{\mathbf{54}}^+$	**69** $^+$	312 $^-$	**174**	205	$\underline{\mathbf{54}}^+$	**78** $^+$
F5	85	$\underline{\mathbf{10}}^+$	**52**	**50** $^+$	**16** $^+$	87	$\underline{\mathbf{10}}^+$	**35** $^+$
F6	178	**48** $^+$	$\underline{\mathbf{45}}^+$	240 $^-$	**104** $^+$	174	$\underline{\mathbf{48}}^+$	**51** $^+$
F7	145	**47** $^+$	$\underline{\mathbf{46}}^+$	226 $^-$	**77** $^+$	142	**47** $^+$	$\underline{\mathbf{44}}^+$
F8	235	**135** $^+$	**92** $^+$	**153** $^+$	$\underline{\mathbf{25}}^+$	366	**135** $^+$	**70** $^+$
F9	165	**68** $^+$	$\underline{\mathbf{67}}^+$	171	**78** $^+$	220	**68** $^+$	$\underline{\mathbf{60}}^+$
F10	172	$\underline{\mathbf{66}}^+$	**110** $^+$	173	**98** $^+$	192	$\underline{\mathbf{66}}^+$	**93** $^+$
F11	149	**52** $^+$	**69** $^+$	**141** $^+$	$\underline{\mathbf{22}}^+$	159	**52** $^+$	**57** $^+$
F12	244	$\underline{\mathbf{64}}^+$	**100** $^+$	**179**	**75** $^+$	297	**64** $^+$	**84** $^+$
F13	178	**54** $^+$	**38** $^+$	**160**	$\underline{\mathbf{25}}^+$	161	**54** $^+$	**26** $^+$
F14	323	**72** $^+$	**209** $^+$	**156** $^+$	$\underline{\mathbf{33}}^+$	361	**72** $^+$	**170** $^+$
F15	166	**64** $^+$	**98** $^+$	**135**	$\underline{\mathbf{18}}^+$	191	**64** $^+$	**72** $^+$
B. **UCI Problems**								
F16	186	$\underline{\mathbf{109}}^+$	**124** $^+$	296 $^-$	**174**	284	$\underline{\mathbf{109}}^+$	**117** $^+$
F17	194	**70** $^+$	$\underline{\mathbf{45}}^+$	198	**84** $^+$	232	**70** $^+$	$\underline{\mathbf{33}}^+$
F18	168	$\underline{\mathbf{74}}^+$	**97** $^+$	340 $^-$	204	220	$\underline{\mathbf{74}}^+$	**86** $^+$
F19	213	**87** $^+$	**13** $^+$	**86** $^+$	$\underline{\mathbf{10}}^+$	317	**87** $^+$	**8** $^+$
F20	240	**92** $^+$	$\underline{\mathbf{91}}^+$	397 $^-$	**212**	331	**92** $^+$	$\underline{\mathbf{86}}^+$
F21	183	$\underline{\mathbf{66}}^+$	**88** $^+$	200	**110** $^+$	237	**66** $^+$	$\underline{\mathbf{58}}^+$
F22	194	**82** $^+$	**84** $^+$	**190**	$\underline{\mathbf{52}}^+$	211	**82** $^+$	**61** $^+$
F23	168	$\underline{\mathbf{52}}^+$	**53** $^+$	233 $^-$	**108** $^+$	212	**52** $^+$	$\underline{\mathbf{20}}^+$
F24	169	**61** $^+$	$\underline{\mathbf{35}}^+$	228 $^-$	**54** $^+$	214	**61** $^+$	$\underline{\mathbf{16}}^+$
F25	174	**70** $^+$	$\underline{\mathbf{34}}^+$	220	**72** $^+$	217	**70** $^+$	$\underline{\mathbf{21}}^+$
F26	137	**37** $^+$	**70** $^+$	**64** $^+$	$\underline{\mathbf{33}}^+$	209	**37** $^+$	**46** $^+$

Table A.3: Mean of best fitness with tour size=5. The left is original data and the right is noise data.

Pro	GP	TS-R	TS-S	TS-P	GP	TS-R	TS-S	TS-P
A. **Benchmarking Problems**								
F1	$\underline{1.59}$	2.50$^-$	2.94$^-$	2.46$^-$	$\underline{1.83}$	2.56$^-$	3.33$^-$	2.50$^-$
F2	$\underline{0.23}$	0.35$^-$	0.58$^-$	0.28$^-$	$\underline{0.21}$	0.37$^-$	0.59$^-$	0.29$^-$
F3	$\underline{4.56}$	6.20$^-$	6.57$^-$	5.08	5.08	5.74$^-$	6.70$^-$	$\underline{\mathbf{4.90}}$
F4	0.05	**0.04**	0.05	$\underline{\mathbf{0.04}}^+$	$\underline{0.10}$	0.11$^-$	0.12$^-$	0.10$^-$
F5	$\underline{0.12}$	0.13	0.13	0.13	0.14	0.14$^-$	0.14	$\underline{\mathbf{0.14}}$
F6	$\underline{0.35}$	0.58$^-$	1.01$^-$	0.56	$\underline{0.61}$	1.02$^-$	1.21$^-$	0.81$^-$
F7	0.42	0.45	0.52$^-$	$\underline{\mathbf{0.41}}$	$\underline{0.46}$	0.49$^-$	0.56$^-$	0.47
F8	5.44	$\underline{\mathbf{4.98}}$	5.48$^-$	**5.01**	$\underline{66.5}$	67.1$^-$	67.2$^-$	66.9$^-$
F9	2.06	**1.73**	2.50$^-$	$\underline{\mathbf{1.39}}^+$	4.15	4.38	5.56$^-$	$\underline{\mathbf{3.96}}$
F10	7.92	**7.47**	$\underline{\mathbf{5.58}}^+$	**7.39**	8.23	8.60	$\underline{\mathbf{6.89}}$	**7.83**
F11	0.09	0.09	$\underline{\mathbf{0.07}}$	**0.08**	0.21	$\mathbf{0.21}^+$	$\underline{\mathbf{0.20}}^+$	**0.21**
F12	$\underline{6.96}$	7.13$^-$	7.07$^-$	7.13$^-$	$\underline{7.02}$	7.16$^-$	7.13$^-$	7.14$^-$
F13	$\underline{0.88}$	0.88$^-$	0.88$^-$	0.88$^-$	$\underline{0.88}$	0.89$^-$	0.90$^-$	0.89$^-$
F14	$\underline{72.8}$	74.3	78.5	77.6	103.6	103.6$^-$	$\underline{\mathbf{102.5}}^+$	$\mathbf{102.7}^+$
F15	2.30	2.50	$\underline{\mathbf{2.11}}$	2.56	$\underline{2.51}$	2.87$^-$	2.62$^-$	2.91$^-$
B. **UCI Problems**								
F16	$\underline{8.08}$	8.78	9.22	8.69	$\underline{18.3}$	20.1$^-$	19.6	18.8
F17	$\underline{3.47}$	4.00$^-$	4.07$^-$	3.80$^-$	$\underline{3.68}$	4.27$^-$	4.35$^-$	4.07$^-$
F18	10.2	11.8	10.4	$\underline{\mathbf{8.9}}^+$	45.3	46.4	$\underline{\mathbf{44.9}}$	45.9
F19	$\underline{25.7}$	29.8$^-$	31.8$^-$	28.3$^-$	$\underline{25.4}$	29.7$^-$	31.6$^-$	28.0$^-$
F20	$\underline{9.36}$	9.84$^-$	9.77	9.58	$\underline{16.4}$	16.7$^-$	16.7$^-$	16.6$^-$
F21	$\underline{4.26}$	4.38$^-$	4.36$^-$	4.30	$\underline{4.40}$	4.50$^-$	4.46	4.48$^-$
F22	$\underline{0.84}$	1.14$^-$	1.10$^-$	1.00$^-$	$\underline{3.25}$	3.69$^-$	3.78$^-$	3.59$^-$
F23	$\underline{3.56}$	4.83$^-$	6.04$^-$	4.23	$\underline{4.18}$	5.51$^-$	7.95$^-$	5.18$^-$
F24	$\underline{8.39}$	10.5$^-$	11.7$^-$	9.74$^-$	$\underline{10.4}$	13.2$^-$	15.2$^-$	12.3$^-$
F25	$\underline{4.57}$	5.69$^-$	6.97$^-$	5.42$^-$	$\underline{5.00}$	6.29$^-$	7.26$^-$	5.94$^-$
F26	$\underline{51.80}$	51.94	52.06	51.88	$\underline{53.11}$	53.35$^-$	53.58$^-$	53.29$^-$

Table A.4: Median of testing error with tour size=5. The left is original data and the right is noise data.

Pro	GP	TS-R	TS-S	TS-P	GP	TS-R	TS-S	TS-P
A. **Benchmarking Problems**								
F1	8.86	**6.07**$^{+}$	$\underline{\mathbf{4.08}}^{+}$	**6.12**$^{+}$	10.9	**6.10**$^{+}$	$\underline{\mathbf{5.17}}^{+}$	**7.90**$^{+}$
F2	0.96	**0.88**$^{+}$	$\underline{\mathbf{0.87}}^{+}$	0.96	0.94	**0.83**$^{+}$	$\underline{\mathbf{0.80}}^{+}$	**0.92**
F3	31.1	**15.3**$^{+}$	$\underline{\mathbf{14.1}}^{+}$	**17.4**$^{+}$	32.4	$\underline{\mathbf{16.1}}^{+}$	**16.2**$^{+}$	**19.3**$^{+}$
F4	0.051	**0.048**	**0.050**	$\underline{\mathbf{0.042}}^{+}$	0.147	**0.143**	**0.143**	$\underline{\mathbf{0.141}}$
F5	0.135	0.135	$\underline{\mathbf{0.129}}$	**0.134**	0.140	**0.140**	$\underline{\mathbf{0.139}}$	0.**140**
F6	$\underline{1.36}$	1.71	1.91	1.92	2.08	2.23	$\underline{\mathbf{2.06}}$	2.23
F7	1.67	1.77	$\underline{\mathbf{1.59}}^{+}$	**1.61**	1.77	1.83	$\underline{\mathbf{1.69}}$	1.81
F8	7.37	**7.26**	7.39	$\underline{\mathbf{6.78}}$	67.1	**66.9**$^{+}$	$\underline{\mathbf{66.8}}^{+}$	**67.0**
F9	1.69	$\underline{\mathbf{1.59}}^{+}$	**1.62**$^{+}$	**1.64**	$\underline{5.16}$	5.49	5.21	5.28
F10	59.7	**48.9**	$\underline{\mathbf{25.4}}^{+}$	**39.7**	61.9	**61.6**	**57.1**	$\underline{\mathbf{56.2}}$
F11	0.07	0.08	$\underline{\mathbf{0.06}}$	0.08	0.199	**0.199**	$\underline{\mathbf{0.198}}^{+}$	0.201
F12	7.44	**7.33**$^{+}$	$\underline{\mathbf{7.33}}^{+}$	**7.37**$^{+}$	7.39	**7.33**	$\underline{\mathbf{7.30}}^{+}$	**7.36**
F13	0.877	**0.874**	$\underline{\mathbf{0.871}}^{+}$	**0.876**	0.90	0.90	$\underline{\mathbf{0.90}}^{+}$	0.90
F14	126.8	127.9	$\underline{\mathbf{124.6}}$	**126.7**	122.7	**122.6**	$\underline{\mathbf{122.5}}^{+}$	**122.7**
F15	4.59	4.99	$\underline{\mathbf{3.58}}$	5.03	4.36	5.00	$\underline{\mathbf{4.13}}$	5.03^{-}
B. **UCI Problems**								
F16	$\underline{21.3}$	22.1	25.3	23.3	37.3	**36.6**	**36.0**	$\underline{\mathbf{34.5}}$
F17	5.12	**4.90**	$\underline{\mathbf{4.71}}^{+}$	**5.03**	5.65	**5.59**$^{+}$	$\underline{\mathbf{5.28}}^{+}$	**5.52**$^{+}$
F18	9.77	10.78	**9.63**	$\underline{\mathbf{6.78}}^{+}$	47.6	**47.4**	$\underline{\mathbf{44.8}}$	**47.0**
F19	40.7	**38.6**$^{+}$	$\underline{\mathbf{36.8}}^{+}$	**39.9**	43.1	**40.3** $^{+}$	$\underline{\mathbf{37.7}}^{+}$	**42.2**
F20	9.59	9.83	$\underline{\mathbf{9.46}}$	9.69	9.32	$\underline{\mathbf{9.13}}^{+}$	**9.14**$^{+}$	**9.18**$^{+}$
F21	4.33	4.36^{-}	4.34	$\underline{\mathbf{4.31}}$	4.51	4.56	$\underline{\mathbf{4.48}}$	**4.57**
F22	1.90	2.14^{-}	**1.82**	$\underline{\mathbf{1.66}}$	5.95	**5.90**	**5.86**	$\underline{\mathbf{5.81}}$
F23	6.84	7.54	8.04	$\underline{\mathbf{6.53}}$	$\underline{7.38}$	7.48	8.48^{-}	8.69
F24	19.1	**16.4**$^{+}$	$\underline{\mathbf{12.8}}^{+}$	**16.5**	24.1	**19.5**$^{+}$	$\underline{\mathbf{16.8}}^{+}$	**22.7**
F25	9.01	**8.51**	**8.33**$^{+}$	$\underline{\mathbf{8.12}}$	9.45	**8.73**	$\underline{\mathbf{8.31}}^{+}$	**8.82**
F26	48.35	**46.95**	$\underline{\mathbf{46.28}}^{+}$	**46.99**	46.64	$\underline{\mathbf{46.51}}$	**46.63**	**46.48**

Table A.5: Average of solution's size with tour size=5. The left is original data and the right is noise data.

Pro	GP	TS-R		TS-S	TS-P		GP	TS-R		TS-S	TS-P	
A. **Benchmarking Problems**												
F1	302	**258**	+	**113**$^+$	**250**	+	292	**245**	+	**106**$^+$	**253**	+
F2	169	**140**	+	**33**$^+$	**164**		174	**148**	+	**29**$^+$	**159**	
F3	277	281		**99**$^+$	**270**		273	274		**104**$^+$	293	
F4	171	205		**70**$^+$	184		270	**219**		**67**$^+$	**228**	
F5	93	**92**		**44**$^+$	110		84	89		**39**$^+$	116	−
F6	164	**146**	+	**56**$^+$	**149**		182	**139**	+	**52**$^+$	**163**	
F7	149	150		**43**$^+$	**137**		138	**137**	+	**58**$^+$	153	
F8	241	**199**	+	**93**$^+$	**201**	+	298	**189**	+	**74**$^+$	**187**	+
F9	209	**141**	+	**70**$^+$	**140**	+	206	**126**	+	**60**$^+$	**139**	+
F10	180	**168**		**102**$^+$	**168**		198	**178**	+	**91**$^+$	**167**	+
F11	157	**145**		**74**$^+$	**149**		156	**144**		**61**$^+$	157	
F12	281	**209**	+	**90**$^+$	**229**	+	292	**212**	+	**86**$^+$	**248**	+
F13	157	**109**	+	**34**$^+$	**148**		172	**141**		**34**$^+$	**147**	+
F14	312	**275**		**171**$^+$	**292**		338	**319**		**156**$^+$	343	
F15	158	**147**		**92**$^+$	159		191	**165**		**79**$^+$	**186**	
B. **UCI Problems**												
F16	227	**226**		**180**$^+$	**215**		250	**234**		**110**$^+$	**219**	
F17	231	**172**	+	**41**$^+$	**186**	+	217	**168**	+	**32**$^+$	**178**	+
F18	198	**198**		**127**$^+$	**182**		195	**175**		**87**$^+$	**183**	
F19	257	**100**	+	**11**$^+$	**171**	+	284	**94**	+	**11**$^+$	**150**	+
F20	240	244		**152**$^+$	**233**		301	**190**	+	**91**$^+$	**215**	+
F21	226	**197**		**89**$^+$	**197**		207	**177**	+	**81**$^+$	**188**	
F22	207	**189**		**87**$^+$	**201**		209	**176**	+	**72**$^+$	**177**	
F23	186	**146**	+	**33**$^+$	**160**		187	**131**	+	**24**$^+$	**147**	+
F24	186	**134**	+	**26**$^+$	**156**	+	201	**121**	+	**20**$^+$	**141**	+
F25	206	**143**	+	**26**$^+$	**159**	+	202	**139**	+	**24**$^+$	**158**	+
F26	220	**201**		**116**$^+$	**218**		171	**147**		**57**$^+$	**143**	

Table A.6: Mean of best fitness of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data.

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RDO
A. **Benchmarking Problems**										
F1	1.59	4.64^{-}	2.94^{-}	$\underline{\mathbf{0.16}}^{+}$	2.29^{-}	1.83	4.78^{-}	3.33^{-}	$\underline{\mathbf{0.14}}^{+}$	3.02^{-}
F2	0.23	0.40^{-}	0.58^{-}	$\underline{\mathbf{0.06}}^{+}$	0.31	0.21	0.41^{-}	0.59^{-}	$\underline{\mathbf{0.06}}^{+}$	0.31
F3	4.56	12.63^{-}	6.57	$\underline{\mathbf{0.16}}^{+}$	**1.06**$^{+}$	5.08	13.11^{-}	6.70	$\underline{\mathbf{0.16}}^{+}$	**1.38**$^{+}$
F4	0.05	0.11^{-}	0.05	$\underline{\mathbf{0.01}}^{+}$	**0.01**$^{+}$	0.10	0.17^{-}	0.12^{-}	$\underline{\mathbf{0.08}}^{+}$	**0.10**
F5	$\underline{0.12}$	0.15^{-}	0.13	0.13	0.15^{-}	$\underline{0.14}$	0.16^{-}	0.14	0.14^{-}	0.15^{-}
F6	0.35	0.77^{-}	1.01^{-}	$\underline{\mathbf{0.01}}^{+}$	**0.01**$^{+}$	0.61	1.00^{-}	1.21^{-}	$\underline{\mathbf{0.28}}^{+}$	**0.58**
F7	0.42	0.50^{-}	0.52^{-}	$\underline{\mathbf{0.19}}^{+}$	**0.40**	0.46	0.54^{-}	0.56^{-}	$\underline{\mathbf{0.25}}^{+}$	0.48
F8	5.44	16.61^{-}	5.48	**0.39**$^{+}$	$\underline{\mathbf{0.37}}^{+}$	66.5	69.2^{-}	67.2^{-}	$\underline{\mathbf{65.8}}$	67.4^{-}
F9	2.06	3.58^{-}	2.50^{-}	$\underline{\mathbf{0.20}}^{+}$	**0.20**$^{+}$	4.15	5.64^{-}	5.56^{-}	$\underline{\mathbf{2.94}}^{+}$	**3.30**
F10	7.92	11.50	**5.58**	**0.95**$^{+}$	$\underline{\mathbf{0.32}}^{+}$	8.23	10.9^{-}	**6.89**	**3.14**$^{+}$	$\underline{\mathbf{2.86}}^{+}$
F11	0.09	0.29^{-}	**0.07**	$\underline{\mathbf{0.03}}^{+}$	**0.06**	0.21	0.30^{-}	**0.20**	$\underline{\mathbf{0.18}}^{+}$	**0.19**$^{+}$
F12	6.96	7.44^{-}	7.07^{-}	$\underline{\mathbf{6.74}}^{+}$	7.04^{-}	7.02	7.52^{-}	7.13^{-}	$\underline{\mathbf{6.74}}^{+}$	7.03^{-}
F13	0.88	0.92^{-}	0.88^{-}	$\underline{\mathbf{0.86}}$	**0.87**$^{+}$	0.88	0.93^{-}	0.90^{-}	$\underline{\mathbf{0.87}}$	0.89^{-}
F14	72.8	83.8^{-}	78.5	$\underline{\mathbf{53.8}}^{+}$	**65.9**$^{+}$	103.6	109.4^{-}	**102.5** $^{+}$	$\underline{\mathbf{96.1}}^{+}$	**103.1**
F15	2.30	3.53^{-}	**2.11**	$\underline{\mathbf{1.10}}^{+}$	**1.11**$^{+}$	2.51	3.95^{-}	2.62^{-}	$\underline{\mathbf{1.87}}^{+}$	**2.02**
B. **UCI Problems**										
F16	8.08	16.73^{-}	9.22	$\underline{\mathbf{2.01}}^{+}$	**2.18**$^{+}$	18.3	23.8^{-}	19.6	$\underline{\mathbf{9.3}}^{+}$	**9.74**$^{+}$
F17	3.47	4.18^{-}	4.07^{-}	$\underline{\mathbf{2.41}}^{+}$	**3.31**	3.68	4.31^{-}	4.35^{-}	$\underline{\mathbf{2.64}}^{+}$	3.71
F18	10.2	26.4^{-}	10.4	$\underline{\mathbf{3.13}}^{+}$	**3.29**$^{+}$	45.3	56.6^{-}	**44.9**	$\underline{\mathbf{34.1}}^{+}$	**35.7**$^{+}$
F19	25.7	28.9^{-}	31.8 $^{-}$	$\underline{\mathbf{23.2}}^{+}$	27.9^{-}	25.4	28.5^{-}	31.6^{-}	$\underline{\mathbf{22.0}}^{+}$	28.5^{-}
F20	9.36	13.5^{-}	9.77	$\underline{\mathbf{6.72}}^{+}$	**7.65**$^{+}$	16.4	16.9^{-}	16.7^{-}	$\underline{\mathbf{14.9}}^{+}$	**15.7**$^{+}$
F21	4.26	4.59^{-}	4.36	$\underline{\mathbf{3.89}}^{+}$	**4.05**$^{+}$	4.40	4.68^{-}	4.46	$\underline{\mathbf{4.01}}^{+}$	**4.17**$^{+}$
F22	0.84	2.37^{-}	1.10^{-}	$\underline{\mathbf{0.55}}^{+}$	**0.71**	3.25	4.22^{-}	3.78^{-}	$\underline{\mathbf{2.75}}^{+}$	3.53^{-}
F23	3.56	6.23^{-}	6.04^{-}	$\underline{\mathbf{0.88}}^{+}$	**2.31**$^{+}$	4.18	7.14^{-}	7.95^{-}	$\underline{\mathbf{1.38}}^{+}$	**3.30**
F24	8.39	11.02^{-}	11.7^{-}	$\underline{\mathbf{3.53}}^{+}$	9.38^{-}	10.4	13.6^{-}	15.2^{-}	$\underline{\mathbf{4.87}}^{+}$	11.4^{-}
F25	4.57	6.43^{-}	6.97^{-}	$\underline{\mathbf{2.07}}^{+}$	**4.62**	5.00	6.79^{-}	7.26^{-}	$\underline{\mathbf{2.09}}^{+}$	5.29
F26	51.80	52.63^{-}	52.07	$\underline{\mathbf{50.88}}$	**51.57**	53.11	53.64^{-}	53.58^{-}	$\underline{\mathbf{52.79}}$	53.24

Table A.7: Median of fittest of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data.

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RDO	TS-RD
A. **Benchmarking Problems**										
F1	8.86	12.59$^-$	**4.08**$^+$	**8.23**	**4.16**$^+$	10.9	13.1	**5.17**$^+$	**10.2**	**6.63**
F2	0.96	**0.84**$^+$	**0.87**$^+$	1.15$^-$	1.00	0.94	**0.84**$^+$	**0.80**$^+$	1.23$^-$	1.00
F3	31.1	32.2	**14.1**$^+$	**4.92**$^+$	**1.85**$^+$	32.4	32.2	**16.1**$^+$	**7.15**$^+$	**6.31**$^+$
F4	0.05	0.12$^-$	0.05	**0.02**$^+$	**0.02**$^+$	0.15	0.19$^-$	**0.14**	**0.14**	**0.14**$^+$
F5	0.135	0.135	**0.129**$^+$	0.138	0.138	0.140	0.140	**0.139**	0.141	0.141$^-$
F6	1.36	1.74	1.91	**0.00**$^+$	**0.00**$^+$	2.08	2.19	**2.06**	3.07	**1.25**$^+$
F7	1.67	**1.61**	**1.59**	**1.22**$^+$	**1.19**$^+$	1.77	**1.73**	**1.69**	**1.61**	**1.62**
F8	7.37	7.41	7.39	**0.00**$^+$	**0.00**$^+$	67.1	**66.9**	**66.8**$^+$	68.5	**66.7**$^+$
F9	1.69	2.41	**1.62**	**0.20**$^+$	**0.23**$^+$	5.16	5.68	5.21	**5.02**$^+$	**4.95**$^+$
F10	59.7	**41.0**	**25.4**	**0.00**$^+$	**0.00**$^+$	61.9	**56.4**	**57.1**	**50.9**$^+$	**46.7**$^+$
F11	0.07	0.30$^-$	**0.06**	**0.00**$^+$	0.08	0.20	0.32$^-$	**0.20**$^+$	**0.20**	**0.20**$^+$
F12	7.44	**7.34**$^+$	**7.33**$^+$	7.49	**7.29**$^+$	7.39	7.41	**7.30**$^+$	7.53$^-$	**7.31**
F13	0.877	**0.874**	**0.871**$^+$	**0.874**	**0.870**$^+$	0.898	**0.898**	**0.896**	0.901	**0.896**
F14	126.8	131.3$^-$	**124.6**	**124.1**	**122.6**$^+$	122.7	128.8$^-$	**122.5**	122.7	**122.6**
F15	4.59	5.92$^-$	**3.58**	**3.24**$^+$	**3.24**$^+$	4.36	6.21$^-$	**4.13**	**4.14**$^+$	**4.12**$^+$
B. **UCI Problems**										
F16	21.3	33.7$^-$	25.3	**6.86**$^+$	**5.86**$^+$	37.3	**36.3**	**36.0**	**12.5**$^+$	**11.5**$^+$
F17	5.12	**4.95**	**4.71**$^+$	5.66$^-$	**4.88**$^+$	5.65	**5.45**	**5.28**$^+$	6.56$^-$	**5.36**
F18	9.77	28.4$^-$	**9.63**	**3.60**$^+$	**3.58**$^+$	47.6	52.9$^-$	**44.8**	**38.6**$^+$	**36.7**$^+$
F19	40.7	**38.3**$^+$	**36.8** $^+$	**37.4**$^+$	**32.2**$^+$	43.1	**40.2**$^+$	**37.7** $^+$	**39.3** $^+$	**35.6**$^+$
F20	9.59	**9.18**	**9.46**	11.7$^-$	11.5 $^-$	9.32	**8.72**$^+$	**9.14**	11.5 $^-$	10.4$^-$
F21	4.33	4.52$^-$	4.34	**4.23**$^+$	**4.18**$^+$	4.51	4.67$^-$	**4.48**	**4.41**	**4.34**$^+$
F22	1.90	3.29$^-$	**1.82**	**1.14**$^+$	**1.18**$^+$	5.95	6.19$^-$	**5.86**	6.02	**5.52**$^+$
F23	6.84	8.44$^-$	8.04	**6.42**	**4.38**$^+$	7.38	9.15$^-$	8.48	10.17$^-$	**5.95**
F24	19.1	**17.7**	**12.8**$^+$	25.2	**14.1**$^+$	24.1	**19.1**$^+$	**16.8**$^+$	27.6	**16.0**$^+$
F25	9.01	**8.89**	**8.33**	15.25$^-$	**7.77**$^+$	9.45	**9.42**	**8.31**$^+$	12.15$^-$	**7.50**$^+$
F26	48.35	**47.26**	**46.28**	**46.35**	**45.11**$^+$	46.64	**46.58**	**46.63**	46.73	46.75

Table A.8: Average of solutions size of TS-RDO and four other techniques with tour size=5. The left is original data and the right is noise data.

Pro	GP	neatGP	TS-S	RDO	TS-RDO	GP	neatGP	TS-S	RI
A. **Benchmarking Problems**									
F1	302	**124**$^+$	**113**$^+$	**227**$^+$	$\underline{\mathbf{62}}^+$	292	**123**$^+$	**106**$^+$	**242**
F2	169	**60**$^+$	$\underline{\mathbf{33}}^+$	**163**	**62**$^+$	174	**65**$^+$	$\underline{\mathbf{29}}^+$	**166**
F3	277	**112**$^+$	**99**$^+$	**161**$^+$	$\underline{\mathbf{48}}^+$	273	**103**$^+$	**104**$^+$	**190**
F4	171	$\underline{\mathbf{60}}^+$	**70**$^+$	336$^-$	178	270	$\underline{\mathbf{54}}^+$	**67**$^+$	336
F5	93	$\underline{\mathbf{12}}^+$	**44**$^+$	**43**$^+$	**15**$^+$	84	$\underline{\mathbf{10}}^+$	**39**$^+$	**37**
F6	164	**45**$^+$	**56**$^+$	**36**$^+$	$\underline{\mathbf{18}}^+$	182	$\underline{\mathbf{48}}^+$	**52**$^+$	234
F7	149	**50**$^+$	$\underline{\mathbf{43}}^+$	207$^-$	**70**$^+$	138	$\underline{\mathbf{47}}^+$	**58**$^+$	224
F8	241	**118**$^+$	**93**$^+$	**13**$^+$	$\underline{\mathbf{10}}^+$	298	**135**$^+$	**74**$^+$	**168**
F9	209	**62**$^+$	**70**$^+$	**69**$^+$	$\underline{\mathbf{35}}^+$	206	**68**$^+$	$\underline{\mathbf{60}}^+$	**190**
F10	180	**60**$^+$	**102**$^+$	**96**$^+$	$\underline{\mathbf{50}}^+$	198	$\underline{\mathbf{66}}^+$	**91**$^+$	**181**
F11	157	**44**$^+$	**74**$^+$	**34**$^+$	$\underline{\mathbf{15}}^+$	156	**52**$^+$	**61**$^+$	**145**
F12	281	**67**$^+$	**90**$^+$	**179**$^+$	$\underline{\mathbf{41}}^+$	292	**64**$^+$	**86**$^+$	**188**
F13	157	**49**$^+$	**34**$^+$	**127**$^+$	$\underline{\mathbf{22}}^+$	172	**54**$^+$	**34**$^+$	**146**
F14	312	**66**$^+$	**171**$^+$	**164**$^+$	$\underline{\mathbf{60}}^+$	338	**72**$^+$	**156**$^+$	**154**
F15	158	**58**$^+$	**92**$^+$	**51**$^+$	$\underline{\mathbf{31}}^+$	191	**64**$^+$	**79**$^+$	**138**
B. **UCI Problems**									
F16	227	$\underline{\mathbf{103}}^+$	**180**$^+$	321$^-$	**172**$^+$	250	$\underline{\mathbf{109}}^+$	**110**$^+$	339
F17	231	**62**$^+$	$\underline{\mathbf{41}}^+$	232	**97**$^+$	217	**70**$^+$	$\underline{\mathbf{32}}^+$	219
F18	198	$\underline{\mathbf{71}}^+$	**127**$^+$	362$^-$	**188**	195	$\underline{\mathbf{74}}^+$	**87**$^+$	392
F19	257	**79**$^+$	**11**$^+$	**85**$^+$	$\underline{\mathbf{8}}^+$	284	**87**$^+$	**11**$^+$	**96**
F20	240	$\underline{\mathbf{87}}^+$	**152**$^+$	374$^-$	**222**	301	**92**$^+$	$\underline{\mathbf{91}}^+$	447
F21	226	$\underline{\mathbf{63}}^+$	**89**$^+$	228	**110**$^+$	207	$\underline{\mathbf{66}}^+$	**81**$^+$	229
F22	207	**83**$^+$	**87**$^+$	**129**$^+$	$\underline{\mathbf{53}}^+$	209	**82**$^+$	**72**$^+$	**194**
F23	186	**55**$^+$	$\underline{\mathbf{33}}^+$	272$^-$	**92**$^+$	187	**52**$^+$	$\underline{\mathbf{24}}^+$	259
F24	186	**68**$^+$	$\underline{\mathbf{26}}^+$	265$^-$	**59**$^+$	201	**61**$^+$	$\underline{\mathbf{20}}^+$	260
F25	206	**63**$^+$	$\underline{\mathbf{26}}^+$	257	**77**$^+$	202	**70**$^+$	$\underline{\mathbf{24}}^+$	248
F26	220	**40**$^+$	**116**$^+$	**54**$^+$	$\underline{\mathbf{29}}^+$	170	**36**$^+$	**57**$^+$	**55**