An Analysis of Selection in Genetic Programming

by

Huayang Xie

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2009

Abstract

This thesis presents an analysis of the selection process in tree-based Genetic Programming (GP), covering the optimisation of both parent and offspring selection, and provides a detailed understanding of selection and guidance on how to improve GP search effectively and efficiently.

The first part of the thesis provides models and visualisations to analyse selection behaviour in standard tournament selection, clarifies several issues in standard tournament selection, and presents a novel solution to automatically and dynamically optimise parent selection pressure. The fitness evaluation cost of parent selection is then addressed and some cost-saving algorithms introduced. In addition, the feasibility of using good predecessor programs to increase parent selection efficiency is analysed.

The second part of the thesis analyses the impact of offspring selection pressure on the overall GP search performance. The fitness evaluation cost of offspring selection is then addressed, with investigation of some heuristics to efficiently locate good offspring by constraining crossover point selection structurally through the analysis of the characteristics of good crossover events.

The main outcomes of the thesis are three new algorithms and four observations: 1) a clustering tournament selection method is developed to automatically and dynamically tune parent selection pressure; 2) a passive evaluation algorithm is introduced for reducing parent fitness evaluation cost for standard tournament selection using small tournament sizes; 3) a heuristic population clustering algorithm is developed to reduce parent fitness evaluation cost while taking advantage of clustering tournament selection and avoiding the tournament size limitation; 4) population size has little impact on parent selection pressure thus the tournament size configuration is independent of population size; and different sampling replacement strategies have little impact on the selection behaviour in standard tournament selection; 5) premature convergence occurs more often

when stochastic elements are removed from both parent and offspring selection processes; 6) good crossover events have a strong preference for whole program trees, and (less strongly) single-node or small subtrees that are at the bottom of parent program trees; 7) the ability of standard GP crossover to generate good offspring is far below what was expected.

Acknowledgments

Big thanks to my thesis supervisors Dr. Mengjie Zhang and Dr. Peter Andreae. Mengjie and Peter have provided much help and stimulated my research, each in a unique way. Mengjie directed my way into the PhD research field and introduced me to the excitement of being an academic researcher, while Peter showed me how to explore my research topics in a wider and deeper fashion and always challenged me to take my research a step further.

Thanks to Victoria University for providing the facility and the scholarship that I need to complete my thesis. In addition, thanks to IEEE CIS, ACM SIGEVO, EuroGP, Marsden Fund, BuildIT, Education New Zealand, Faculties of Science at Victoria University, and Victoria University Research Fund for providing me with a travel fund to enable me attend valuable international conferences.

Great thanks extend to a few peers. Dr. Neale Ranns provided me with his wonderful GP package (Gouda) and endless technical support. Dr. Ivy Liu, Dr. Mark Johnston and Simon Doherty always had their doors open to me for any discussion. Dr. Val Hopper always provided me with a lot of encouragement. Great thanks also go to people not mentioned here who have contributed to my research, including anonymous reviewers, our GPEC research group members, and the technical group in our school.

Last but not least, special thanks to my family for their consistent support and understanding. During the last three years, they have provided me with a stable environment and have taken over the majority of my family commitments to enable me focus on my research.

Publications Produced

In the course of my Ph.D study, I produced several fully-refereed publications, most of which have been further improved and included in the thesis.

Huayang Xie, Mengjie Zhang, and Peter Andreae. "An Analysis of the Distribution of Swapped Subtree Size in Tree-based Genetic Programming". In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2864–2871, IEEE Computer Society Press (2008).

Huayang Xie, Mengjie Zhang, Peter Andreae, and Mark Johnston. "An Analysis of Multi-Sampled Issue and No-Replacement Tournament Selection". In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1323–1330, ACM Press (2008).

Huayang Xie, Mengjie Zhang, Peter Andreae, and Mark Johnston. "Is the Not-Sampled Issue in Tournament Selection Critical?". In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 3711–3718, IEEE Computer Society Press (2008).

Peter Andreae, **Huayang Xie**, and Mengjie Zhang. "Genetic Programming for Detecting Rhythmic Stress in Spoken English". *International Journal of Knowledge-Based and Intelligent Engineering Systems, Special Issue on Genetic Programming*, Volume 12, Number 1, pages 15–28, (2008).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "An Analysis of Depth of Crossover Points in Tree-based Genetic Programming". In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 4561–4568, IEEE Computer Society Press (2007).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "Genetic Programming for New Zealand CPI Inflation Prediction". In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2538–2545, IEEE Computer Society Press (2007).

Huayang Xie, Mengjie Zhang and Peter Andreae. "Another Investigation on Tournament Selection: modelling and visualisation". In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1468–1475, ACM Press (2007).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "An Analysis of Constructive Crossover and Selection Pressure in Genetic Programming". In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1739–1746, ACM Press, (2007).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "Automatic Selection Pressure Control in Genetic Programming". In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, pages 435–440, IEEE Computer Society Press, (2006).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "A Study of Good Predecessor Programs for Reducing Fitness Evaluation Cost in Genetic Programming". In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 9211–9218, IEEE Computer Society Press (2006).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "Population Clustering in Genetic Programming". *Lecture Notes in Computer Science*, Volume 3905, pages 190–201, Springer-Verlag (2006).

Huayang Xie, Mengjie Zhang, and Peter Andreae. "Genetic Programming for Automatic Stress Detection in Spoken English". *Lecture Notes in Computer Science*, Volume 3907, pages 460–471, Springer-Verlag (2006).

Huayang Xie. "Diversity Control in GP with ADFs for Regression Tasks". *Lecture Notes in Artificial Intelligence*, Volume 3809, pages 1253-1257, Springer-Verlag (2005).

Contents

List of Tables

List of Figures

Chapter 1

An Analysis of Selection

This thesis investigates issues of selection in tree-based Genetic Programming (GP) [93]. GP is a powerful program-induction and search methodology. It is a form of Evolutionary Algorithms (EAs) based on the Darwinian natural selection theory. It searches for computer programs to solve a given problem without being explicitly told how. There are many factors affecting the performance of EAs. Selection is a key element because other factors, including population diversity, are the consequential factors of it. How to select states and make movements in a search space is an important issue in order to build an evolutionary algorithm to solve a given problem effectively and efficiently. This thesis describes a series of carefully-designed experiments and analyses that provide a detailed understanding of the selection behaviour in a tree-based GP search process and give guidance on how to improve the effectiveness and efficiency of the GP search.

1.1 Introduction

GP started to receive attention from a wide group of researchers from the early 1990s. Since then, it has been rapidly developed into a popular research field of artificial intelligence. GP has been recognised as being able to find promising solutions in many areas, including signal filters [6, 21, 141], circuit designing [37, 95, 151], image recognition [2, 3, 188], symbolic regression [26, 171, 162], financial prediction [103, 106, 217], and classification [76, 212, 213].

To fulfill a certain task, GP starts with a randomly-initialised population of

programs. It evaluates each program's performance using a fitness function, which generally compares the program's outputs with the target outputs on a set of training data ("fitness cases"). It assigns each program a fitness value, which in general represents the program's degree of success in achieving the given task. Based on the fitness values, it then chooses some of the programs using a stochastic selection mechanism, which consists of a selection scheme and a selection pressure[1] control strategy. After that, it produces a new population of programs for the next generation from these chosen programs using crossover (sexual recombination), mutation (asexual), and reproduction (copy) operators. The search algorithm repeats until it finds an optimal or acceptable solution, or certain stopping criteria are met.

GP search can have two extremes [183] according to the configuration of selection pressure. One extreme, when there is no selection pressure, is completely stochastic so that the GP search acts just like the Monte Carlo method [159], randomly sampling the space of feasible solutions. The other extreme, when the selection pressure is very high, is minimally stochastic so that the GP search acts like a local hill-climbing search method. It is clear that in general the drawback of the former extreme is its inefficiency and the drawback of the latter extreme is its possible confinement to local optima or "*premature convergence*". Therefore, an effective and efficient GP search algorithm must balance between these two extremes. In order to obtain the balanced situation, selection pressure, the key element in the selection mechanism, must be properly controlled so that the stochastic elements are maintained at an optimal level.

Selection in GP search consists of parent selection and offspring selection. The selection of parents has been well explored through the history of the development of EAs. Selection pressure is applied to the parent selection process to reduce the stochastic element of the GP search and to provide individuals having good fitness with more chances to be chosen as parents than others. Good genetic material in the chosen parents is expected to be propagated along evolution in order to speed up population convergence. There is a variety of selection schemes for parent selection. These selection schemes have their own advan-

[1] It can be seen as a bias in favour of the fitter individuals. Detailed definition and explanations can be found in Chapter 2 Section 2.4 on page 17.

tages and drawbacks, as well as different flexibilities of the selection pressure control. However, many GP related research projects and applications either use a selection mechanism based on empirical search or simply follow others without sufficient justification [5, 7, 115, 181]. Some alternative selection mechanisms were developed in order to improve originals but their effectiveness was demonstrated via some experimental results without in-depth analyses [142, 172]. Lack of understanding of the working of a selection scheme impedes addressing and eliminating its existing drawbacks, as well as properly manipulating its selection pressure. Further investigation of parent selection mechanisms is necessary.

The selection of offspring (choosing which offspring to put into the next generation) was effectively missing in GP search originally because the creation of offspring was a random process without selection pressure and created offspring were directly put into the next generation, meaning that "Survival of the fittest" was not applied to offspring. Recently, researchers noticed that the number of possible offspring in the immediate neighbourhood of any chosen parents is large, and a large fraction of these offspring will not constitute improvement over the parents [133, 134]. Applying selection pressure to the offspring selection process was therefore suggested. The selection of offspring has received more attention as shown by a large number of attempts to develop new genetic operators. However the effectiveness of the use of offspring selection is still under investigation. It is not clear whether further reducing the stochastic element of the GP search in the offspring selection process will result in premature convergence or other undesirable restrictions on the GP search.

1.2 Goals

The major goal of this thesis is to analyse selection behaviour in tree-based GP in order to understand the requirements for an effective and efficient selection mechanism and develop improved selection mechanisms, as well as provide useful guidance on constructing further improved selection mechanisms.

The first part of the thesis analyses the parent selection behaviour and investigates the following research:

- how parent selection pressure should be properly controlled; and
- how the cost of fitness evaluation in the parent selection process can be minimised.

The second part of the thesis analyses the impact of offspring selection and investigates the following research:

- how applying offspring selection together with a parent selection mechanism affects GP search results;
- how parent selection pressure and offspring selection pressure should be configured in order to significantly improve the effectiveness of GP search; and
- how the exploration of good offspring search space can be constrained structurally in order to minimise the fitness evaluation cost in the offspring selection process.

1.3 Major Contributions

The thesis makes the following major contributions:

1. An analysis of selection behaviour in one of the most commonly-used parent selection method — tournament selection — in GP shows that in order to significantly improve parent selection, the key point is to tune parent selection pressure automatically and dynamically along evolution, which can be done by integrating the characteristics of a population into the tournament selection, instead of using different sampling replacement strategies.

   Part of this work was published in [200, 201, 204, 206, 207].

2. An analysis of the impact of offspring selection on the overall GP performance shows that increasing offspring selection pressure can improve GP search performance but premature convergence occurs more often if parent selection pressure is not reduced accordingly. It is preferable to apply selection pressure to offspring selection rather than to the commonly-recognised parent selection.

Part of this work was published in [202].

3. An analysis of program structure in good crossover events shows that the exploration of good offspring search space for crossover can be constrained using a combination of unequal-probability depth and small subtree size strategies.

   Part of this work was published in [203, 205].

1.4 Thesis Outline

The structure of the thesis is the following:

- Chapter 2 briefly reviews machine learning, evolutionary algorithms, and the components of GP. It then focuses on evolutionary paradigm independent parent selection mechanisms and tree-based GP specified crossover operators to review ways of improving the effectiveness and efficiency of GP search.

- Chapter 3 focuses on one of the most commonly-used selection scheme in GP — tournament selection — to show how parent selection pressure should be tuned along the evolutionary process.

- Chapter 4 investigates ways to improve the efficiency of tournament selection for parents.

- Chapter 5 focuses on crossover and investigates the impact of offspring selection pressure on the overall GP performance as well as the selection pressure configuration between parent and offspring selections.

- Chapter 6 continues to focus on crossover to investigate heuristics for improving the efficiency of searching good offspring by analysing some properties of good crossover points.

- Chapter 7 draws conclusions and presents directions for future research.

Chapter 2

Literature Review

This chapter starts with introducing machine learning, then evolutionary algorithms, followed by a brief review of components of GP. It then reviews a generic and evolutionary algorithm paradigm independent element — parent selection mechanism — together with a closely-related open issue, fitness evaluation cost. Finally it focuses on genetic operators, especially the crossover operator in tree-based GP, to review ways of improving the effectiveness and efficiency of GP search, particularly integrations of local search metaphors and strategies of controlling positions of possible crossover points.

2.1 Machine Learning

Machine learning is one of the hottest research areas of artificial intelligence. It has been widely adopted in many real-world applications, including natural language processing [117], hand-written character recognition [116], bioinformatics [13], search engines [105], and robot locomotion [38]. The major focus of machine learning research is to extract information from data sources automatically, by computational and statistical methods [11, 65].

2.1.1 Definitions

Researchers give different definitions of machine learning. However the principle is roughly the same: a computer program processes a given set of examples and

tries to either describe the known data source in some meaningful ways or develop an appropriate response to unseen cases. Three representative definitions or descriptions of machine learning are listed below:

Mitchell [126] gives the following definition of machine learning:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improve with experience E."

Witten and Frank [197] state that:

> "... things learn when they change their behavior in a way that makes them perform better in the future ..."

Michalski *et al.* [121] state that:

> "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

2.1.2 Learning data source

A data source is a collection of examples. A single item in a data source is called an *instance*. There are one or more *attributes* or *features* representing the aspect(s) of an instance. Each attribute can have either a categorical or a numerical value.

In order to conduct training and evaluate the performance of a solution, the data source is usually split into two subsets: *a training data set* and *a test data set*. The training data set is used to induce an algorithm to learn and the test data set is used to evaluate how well the algorithm has learned. Sometimes it is split into three subsets. The third data set is usually called *validation data set*. The purpose of using a validation data set is to monitor the training progress and prevent the training from overfitting. When the data set is too small, the *n-fold cross validation* method is used [126]. The m available examples are randomly partitioned into n disjoint subsets, each of size m/n. Training and testing processes are then run n times. In each run, a different one of these n subsets is used as the test data set and all the other subsets are merged and used as the training data set. The averaged test result is reported as the system performance.

2.1.3 Learning paradigms

According to [85], based on the knowledge provided, there are three main learning paradigms: *supervised learning*, *unsupervised learning*, and *hybrid learning*. Supervised learning is sometimes referred to as learning with a teacher. The knowledge provided to a learning system includes a correct answer for each input instance. The learning process is continued until the learning system produces answers as close as possible to the given correct answers. Unsupervised learning is sometimes referred to as learning without a teacher. Instances are grouped into appropriate categories by analysis. A typical problem dealt with by unsupervised learning is *clustering* [49, 67]. "Hybrid learning combines both supervised and unsupervised learning. Part of the solutions (network weights, architecture, or computer programs) are determined through supervised learning, while the others are obtained through unsupervised learning." [85]

There are many machine learning methods in common use, including Bayesian inference [196], decision trees [154], neural networks [130], support vector machines [32], and evolutionary computation.

2.2 Evolutionary Computation

Evolutionary computation is a subfield of artificial intelligence. Often it is inspired by biological mechanisms of evolution and uses iterative and parallel processing to search solutions. It mainly comprises evolutionary algorithms and swarm intelligence.

2.2.1 Evolutionary algorithms

Evolutionary Algorithms (EAs) are inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest, that is, the Darwinian natural selection theory. "Survival of the fittest" is the familiar concept known to drive evolution. What is meant by *survival* in a qualitative and quantitative sense? The answer is that a *genotype*[1] survives across

[1]The internally coded, inheritable information carried by all living organisms.

generations through the production of offspring, and the production of offspring is coupled to the fitness of the corresponding *phenotype*[2]. In nature this coupling may be achieved by the ability of an individual to defeat its competitors of the same species in a contest for mating, or by the ability of the individual to obtain more food or to run and/or hide from predators of other species in order to live long enough to mate. More often, survival is a combination of many factors, with only one in common across all species and environments: random chance.

An idea to use Darwinian natural selection theory for automated problem-solving originated in the 1950s [43]. Since the 1960s, three distinct interpretations of the idea started to be developed in three different places: Evolutionary Programming (EP) was introduced by Lawrence J. Fogel [50]; Genetic Algorithms (GAs) was introduced by John Henry Holland [74]; and Evolution Strategies (ES) was introduced by Ingo Rechenberg and Hans-Paul Schwefel [155]. Later on, in the early 1990s, a fourth stream, genetic programming [93] — a specialisation of GAs — has emerged.

In addition to being categorised according to actual search techniques, EAs can also be categorised based on how population is replaced. In general population size is kept constant. When reproduction takes place, new individuals must replace existing individuals. If the number of individuals created is equal to the population size, then the entire population is replaced and an entire generation is created. This approach is hence referred to as generational EAs. If on the other hand, the number of individuals replaced is actually quite small, for example one, then the new population is actually a mix of new and old generations. This approach is called steady state EAs [101]. Researchers have done several studies on their impact comparisons [45, 87, 189].

Since EAs consist of populations of individuals that produce offspring via a variety of reproductive mechanisms which introduce genetic variation into the population, it is possible that changing some parameters, including the population size, the type and amount of reproductive variation, could significantly change the search behaviour of an EA on a particular fitness landscape [36]. There have been a large number of research studies on different methods to automat-

[2]The outward, physical manifestation of a living organism, including parts of the observable structure, functions and behaviour.

ically change these parameters as well as understand their interactions for improving the performance of EAs. A comprehensive collection of these studies can be found in [108].

2.2.1.1 genetic algorithms

In GAs, a candidate solution is called an *individual* or a *chromosome*. For a given problem, variables are encoded into a specified representation, for instance, a string of binary digits, a string of integers, or a string of floating-point numbers, and each variable is termed as *gene*. It is important to choose the "right" representation for a given problem. Getting the representation right is one of the most difficult parts of designing a good genetic algorithm. Through selection and recombination genetic material is exchanged among individuals, building blocks[3] are expected to be constructed, and finally an acceptable solution is expected to be found. In addition to the right representation, whether the search of a genetic algorithm is successful depends on the choice and the probability configurations of genetic operators, population size, and number of generations etc. Further information on GAs can be found in [125].

2.2.1.2 evolution strategies

In general, ES is used for continuous parameter optimisation. Its representations are real valued vectors and do not need an encoding step to map its genotype space to its phenotype space. ES heavily uses mutation operators, which are based on Gaussian distribution and can self-adapt the step sizes to evolve individuals. It selects parents randomly from a population of size μ and combines two parents to produce only one child. It has two offspring selection methods. After creating λ children, the best μ of them are chosen based on fitness, either from the λ offspring only, called (μ, λ) selection, or from the union of parents and children, called $(\mu + \lambda)$ selection. Further information on ES can be found in [16].

[3] "Building-blocks" are useful sub-components of an individual. This concept has been discussed and studied widely. However, there is no universally agreed definition of what kinds of sub-components can be building-blocks in different EAs.

2.2.1.3 evolutionary programming

EP was originally developed to simulate evolution as a learning process aiming to generate artificial intelligence [43]. Finite state machines were originally used to represent predictors but nowadays floating-point vectors are used very often. EP treats every individual in a population as part of a specific species rather than as different members of the same species. Therefore, EP does not have parent selection pressure and recombination operators. Every individual in a population is mutated to produce one child. EP selects μ individuals into the next generation from the union of parents and offspring (a $(\mu + \mu)$ method). Further information on EP can be found in [43].

2.2.1.4 other evolutionary algorithms

learning classifier systems Learning Classifier Systems (LCS) [75] have a close relationship between reinforcement learning and genetic algorithms. Initially, LCS consisted of a population of binary rules whose fitness was based on a reinforcement learning technique while individuals were evolved by a genetic algorithm. Recently, research has expanded the representation to include real-valued, neural network, and functional conditions. According to where a genetic algorithm acts, LCS can be categorised into two styles: *Pittsburgh* and *Michigan*. A Pittsburgh-style LCS has a population of separate rule sets, where the genetic algorithm recombines and reproduces the best of these rule sets. A Michigan-style LCS has only a population of a single rule set where the genetic algorithm focuses on selecting the best classifiers within that rule set. In general, "learning classifier system" refers to Michigan-style LCS.

differential evolution Differential Evolution (DE) [176] grew out of Kenneth Price's attempts to solve the Chebychev polynomial fitting problem using vector differences for perturbing the vector population. DE is a very successful method of using the differential mutation for global optimisation over continuous spaces. The significant difference between DE and other evolutionary algorithms is its scheme for generating trial parameter vectors. DE adds a weighted difference between two population vectors to a third vector. By doing this, no separate

probability distribution is required so that the scheme becomes completely self-organising (see [152] for details).

estimation of distribution algorithms Estimation of Distribution Algorithms (EDA) [128] are an outgrowth of genetic algorithms. In order to avoid the disruptions of building blocks or partial solutions, EDAs do not use crossover or mutation operators to produce the next generation. Instead they generate the new population by sampling the probability distribution, which is estimated from selected individuals of the previous generation and is supposed to characterise the distribution of promising solutions.

For a given problem, in general, all the variables in the problem are assumed to be independent so that the estimation of the probability distribution from selected individuals can be easily calculated. However, in many real world problems this assumption seldom holds, leaving how to estimate the probability distribution from selected individuals as a hot research topic.

2.2.2 Swarm intelligence

Swarm Intelligence (SI) [90] is inspired by many natural examples, including ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling. An SI system is typically made up of a population of individuals interacting with each other and with the environment following simple rules, thus having a collective, decentralised, and self-organised global behaviour. SI is mainly used for optimisation problems. It incorporates many techniques, including ant colony optimisation [41], particle swarm optimisation [89], and bacterial foraging optimisation [140].

2.3 GP — A Genetic Search Process

Genetic programming is a technique enabling a genetic algorithm to search a potentially infinite space of computer programs, rather than a space of fixed-length solutions to a combinatorial optimisation problem. These programs often take the form of Lisp symbolic expressions, called *S-expressions*. The idea of applying GAs

to S-expressions rather than combinatorial structures originated with Stephen F. Smith [170], and was brought to prominence through the work of Koza [93]. The S-expressions in GP correspond to programs which a user seeks to adapt to perform pre-specified tasks. The fitness of an S-expression may therefore be evaluated in terms of how effectively it performs this task [88]. GP with individuals in S-expressions is referred to as tree-based GP.

GP also has other categories based on the representations of an individual, for instance, linear structure GP [14, 47, 135, 136], graph-based GP [62, 70, 124, 143] and grammar-based GP [193, 59, 137, 198]. Linear structure GP is based on the principle of register machines, thus programs can be linear sequences of instructions. Graph-based GP is suitable for the evolution of highly parallel programs which effectively reuse partial results. Grammar-based GP uses a context free grammar to define the initial GP structures and restrict crossover and mutation operations in order to ensure legal programs are always created.

A much more comprehensive field guide to GP can be found in [147]. The rest of this section reviews GP in a simple and straightforward way.

2.3.1 Generating an initial population

There are three very common methods in tree-based GP to generate an initial population of programs for GP to use when starting the genetic search process. They are the *grow* method, the *full* method, and the *ramped half-and-half* method [93].

Grow method When the grow method is used, each tree of the initial population is built using the following algorithm:

1. a random symbol is selected with uniform probability from the function set to be the root of the tree;

2. if n is the arity of the selected function symbol of a node, then n symbols are selected with uniform probability from the function set and the terminal set to be the node's sub-nodes;

3. for each of the n sub-nodes where its symbol is a function, the method is recursively applied from (2) unless the depth of the sub-node reaches the predefined limit. Then its sub-nodes will be selected only from the terminal set.

In the Grow method, the root is selected with uniform probability from the function set, so that no tree is composed by a single node initially. Given the maximum tree depth d, nodes with depths between 1 and d-1 are selected with uniform probability from the function and the terminal set, but once a branch contains a terminal node that branch will be terminated, even though the maximum tree depth has not been reached. Finally, nodes at depth d are chosen with uniform probability from the terminal set.

Since the incidence of choosing sub-nodes from the function set and the terminal set is random throughout the initialisation process[4], trees are likely to have irregular shape, containing branches of various lengths.

Full method Instead of selecting sub-nodes from the function set and the terminal set, the full method creates sub-nodes by choosing a function symbol only from the function set until the tree depth reaches the predefined limit. Then it terminates the tree by choosing a terminal from the terminal set. The method guarantees that every branch of a tree has the maximum depth.

Ramped half-and-half method In order to enhance the diversity in the initial population, the ramped half-and-half method was introduced in [93] to reduce the chance of having very similar programs (programs too close to each other) as in the previous two methods. Let d be the predefined maximum tree depth. The population is divided evenly among programs to be initialised with trees having depths equal to 1, 2, ..., d-1, d. For each depth group, half of the trees are created using the grow method, and the other half using the full method.

[4]Some GP implementations [165] have biased the selection of functions during subtree generation toward those with a higher arity.

2.3.2 Evaluating programs

A program needs to be evaluated so that its quality of solving a given problem can be represented. An explicit evaluation function to evaluate programs is called *fitness function*, and *fitness* nowadays often refers to the result of the evaluation, differing from its original meaning. In almost all research of GP, fitness is used to determine whether a solution has been found and to select programs for producing the next generation.

There are several measurements of fitness. "*Raw fitness* is the measurement of fitness that is stated in the natural terminology of the problem itself" [93]. It has been widely interpreted as a performance measure. Koza also described three other alternative measurements of fitness based on the raw fitness, including *standardised fitness*, *adjusted fitness*, and *normalised fitness*. The standardised fitness is used when a better fitness needs to be represented in a lower numerical value. The adjusted fitness and the normalised fitness are used mainly in fitness-proportional selection (see Section 2.4.2.1 on page 25 for details) but are not relevant in ranking selection and tournament selection.

The design of the fitness function is critical. For example, for deriving a program to fire a gun, only knowing whether the target has been hit is not a helpful fitness function to guide the next fire to achieve a better result. While knowing which direction the target was missed can be more helpful, it is still not good enough for the search process to work out exactly how much should be adjusted for the next time. But if both the direction and the distance by which the bullet misses its target are used as the fitness function, the search will be much more effective and the target will be easily hit.

The fitness function can be a single objective method or a multi-objective method where multiple program properties, including correctness, parsimony, and efficiency [93, 111, 112], are considered. A single objective fitness function outputs a single fitness value while a multi-objective fitness function can output a single fitness value by combining multiple weighted values together, or producing a vector of values. Recently, dynamic fitness functions and hierarchically-defined fitness functions were proposed to guide GP search [100].

It is relatively easy to define a meaningful fitness function when the problem is

already mathematically formulated. For problems, like robot control, translating a desired behaviour into an objective function is a formidable task. Therefore, Tettamanzi [184] introduced a selection scheme based on the idea of competition in order to evaluate programs without explicitly defining a fitness function.

2.3.3 Generating next generation

GP needs to explore the next generation of programs to carry on the genetic search. The way GP locates the next population is through the processes of selecting parents and producing offspring. It selects programs according to their fitness and applies genetic operators, including reproduction, mutation, and crossover, to the selected programs for generating offspring. There is much research on selection strategies and genetic operators. A detailed overview of this can be found in the next sections, 2.4 and 2.6.

2.4 Parent Selection

A critical issue in the design of a selection technique is *selection pressure*. Many definitions of selection pressure can be found from the literature. For instance, it is defined as the intensity with which an environment tends to eliminate an organism and thus its genes, or gives it an adaptive advantage [80], or as the impact of effective reproduction due to environmental impact on the phenotype [35], or as the intensity of selection acting on a population of organisms or cells in culture. These definitions originate from different perspectives but they share the same aspect, which can be summarised as the degree to which the better individuals are favoured [122]. Selection pressure gives individuals of higher quality a higher probability of being used to create the next generation so that EAs can focus on promising regions in the search space [18].

Selection pressure controls the selection of individual programs from the current population to produce a new population of programs in the next generation. This is important in a genetic search process because it directly affects the population convergence rate. The higher the selection pressure, the faster the convergence. A fast convergence decreases learning time, but often results a GP

learning process being confined in a local maximum or "*premature convergence*" [31, 93]. A slow convergence rate generally decreases the chance of premature convergence but also increases the learning time and may not be able to find an optimal solution in a preset limited time.

There is a wide range of selection techniques in EAs. There are mainly three types of selection methods to select candidates for producing offspring in GP. They are *tournament selection*, *fitness-proportional selection*, and *ranking selection*.

2.4.1 Tournament selection

According to the description given by Goldberg and Deb [57], the initial study of tournament selection can be traced back to the early 1980s [23]. One form of the conventional tournament selections introduced in [23] has became the *standard*.

The standard tournament selection randomly samples k individuals with replacement[5] from the current population of size N into a tournament of size k and selects the one with the best fitness from the tournament[6].

By using different tournament sizes, the selection pressure can be changed to influence the convergence of the genetic search process. In general, the larger the tournament size, the higher the selection pressure. However, when population starts to converge, many programs have the same fitness value, the selection behaviour in standard tournament selection starts to become random [60]. Therefore, tournament size is not always an adequate measure of selection pressure.

2.4.1.1 selection pressure measurements

In tournament selection, the mating pool consists of winners. The average fitness in the mating pool is higher than that in the population. The fitness difference between the mating pool and the population reflects the selection pressure, which is expected to improve the fitness of each succeeding generation [122].

In biology the effectiveness of selection pressure can be measured in terms of differential survival and reproduction, and consequently in the change in the

[5]This can be viewed as making a copy of an individual for a tournament, thus the population remains unchanged.

[6]Some implementations return two individuals if the tournament size is far bigger than two, for instance, the GP C++ Class Library [192].

frequency of alleles in a population. In EAs, from the literature, there are several measurements for selection pressure in different contexts, including *takeover time, selection intensity, loss of diversity*, and *reproduction rate*.

Takeover time is introduced by Goldberg and Deb [57] to quantify the selection pressure. It is defined as the number of generations required to completely fill a population with just copies of the best individual in the initial generation when only selection and copy operators are used. For a given fixed-sized population, the longer the takeover time, the lower the selection pressure. Goldberg and Deb estimated the takeover time for standard tournament selection using the asymptotic expression

$$\frac{1}{\ln k}\left(\ln N + \ln(\ln N)\right) \tag{2.1}$$

where N is the population size and k is the tournament size. The approximation improves when $N \to \infty$. However, this measure is static and constrained and therefore does not reflect the selection behaviour dynamics from generation to generation in EAs.

Selection intensity is another measure for selection pressure. This was firstly introduced in the context of population genetics to obtain a normalised and dimensionless measure [24], and, later was adopted and applied to GAs [129]. Blickle and Thiele [18, 19] measured it using the expected change of the average fitness of the population. As the measurement is dependent of the fitness distribution in the initial generation, they assumed the fitness distribution followed the normalised Gaussian distribution and introduced an integral equation for modelling selection intensity in standard tournament selection.

For their model, analytical evaluation can be done only for smaller tournament sizes and numerical integration is needed for a larger tournament size. The model is not valid in the case of discrete fitness distributions. In addition to these limitations, the assumption that the fitness distribution followed the normalised Gaussian distribution is not valid in general [150]. Furthermore, because the actual fitness values are ignored but the relative rankings are used in tournament selection, the model is of limited use.

Loss of diversity is defined as the proportion of individuals in a population that are not selected during the selection phase [18, 19]. Blickle and Thiele [18, 19] estimated the loss of diversity in the standard tournament selection as:

$$k^{-\frac{1}{k-1}} - k^{-\frac{k}{k-1}} \tag{2.2}$$

However, Motoki [127] pointed out that Blickle and Thiele's estimation of the loss of diversity in tournament selection does not follow their definition, and indeed their estimation is of loss of *fitness* diversity. Motoki recalculated the loss of *program* diversity in a *wholly diverse* population , i.e., every individual has distinct fitness value, on the assumption that the worst individual is ranked 1st, as:

$$\frac{1}{N}\sum_{j=1}^{N}(1 - P(W_j))^N \tag{2.3}$$

where $P(W_j) = \frac{j^k-(j-1)^k}{N^k}$ is the probability that an individual of rank j is selected in a tournament.

"Reproduction rate" is defined as the ratio of the number of individuals with a certain fitness f after and before selection [18, 19]. A reasonable selection method should favour good individuals by giving them a high ratio and penalise bad individuals by giving a low ratio. Branke *et al.* [22] introduced a similar measure which is the expected number of selections of an individual. It is calculated by multiplying the selection probability of the individual in a single tournament by the total number of tournaments conducted in the selection phase. This measure is termed *selection frequency* in this thesis hereafter rather than reproduction rate, which has another meaning in GP. Branke *et al.* provided a model to calculate the selection frequency for a single individual of rank j in the standard tournament selection in a wholly diverse population on the assumption that the worst individual is ranked 1st, as:

$$N\frac{j^k - (j-1)^k}{N^k} \tag{2.4}$$

2.4.1.2 models of sampling behaviour and selection behaviour in tournament selection

There are many papers modelling and comparing the selection behaviour of a variety of selection schemes [15, 19, 22, 57, 123, 127]. There are also many dedicated studies on standard tournament selection [18, 122, 146].

Based on the concept of takeover time [57], Bäck [15] compared several selection schemes, including tournament selection. He presented the selection probability of an individual of rank j in one tournament for a minimisation task[7], with an implicit assumption that the population is wholly diverse, as:

$$N^{-k}((N-j+1)^k - (N-j)^k) \tag{2.5}$$

In order to model the expected fitness distribution after performing tournament selection in a population with a more general form, Blickle and Thiele extended the selection probability model in [15] to describe the selection probability of individuals with the same fitness. The model, though elegant, is somewhat abstract. They defined the worst individual to be ranked 1st and introduced the *cumulative fitness distribution*, $S(f_j)$, which denotes the number of individuals with fitness value f_j or worse. They then calculated the selection probability of individuals with rank j as:

$$\left(\frac{S(f_j)}{N}\right)^k - \left(\frac{S(f_{j-1})}{N}\right)^k \tag{2.6}$$

In order to demonstrate the computational savings in backward-chaining evolutionary algorithms, Poli and Langdon [146] calculated the probability that one individual is not sampled in one tournament as $1 - \frac{1}{N}$, then consequently the expected number of individuals not sampled in any tournament as:

$$N\left(\frac{N}{N-1}\right)^{-ky} \tag{2.7}$$

where y is the total number of tournaments required to form an entire new generation.

[7] Therefore the best individual is ranked 1st.

There is also some experimental work on analysing the selection behaviour of tournament selection. Gathercole [53] showed the selection frequency of each individual and the likelihoods of not-selected and not-sampled individuals in tournament selection of different tournament sizes through 1000 simulations on a sample population size of 50. In his simulation, only one child is produced by crossover or mutation, thus the total number of tournaments required to generate the next entire population is a function of the crossover rate, the mutation rate and the population size, instead of being the same as the population size. His experimental results are interesting, however it is not clear in his simulation whether sampling tournament candidates was done with or without replacement and whether the sample population was fully diverse or not.

2.4.1.3 variations based on standard tournament selection

Some interesting but by no means complete alternative tournament selection implementations are briefly reviewed below.

An alternative tournament selection that can tune selection pressure at a fine level was presented in [57]. In the form of tournament selection, an extra probability p is introduced. When conducting a tournament between two individuals, the individual with higher fitness value can be selected as a parent with the probability p, while the other has the probability $1 - p$. By setting p between 0.5 and 1, it is possible to tune the selection pressure continuously between the random selection and the tournament selection with tournament size two. Recently, Hingee and Hutter [69] showed that every probabilistic tournament is equivalent to a unique polynomial ranking selection scheme.

Harik [63] demonstrated some interesting work in tournament selection in the context of GAs. He introduced a restricted tournament selection method in GAs for two purposes. The first was to preserve and find multiple solutions and the second was to obtain a particular global solution by taking advantage of the schema found in multiple local solutions. In the restricted tournament selection, two parents are randomly selected from a population to produce two offspring. For each offspring, a number of competitors are randomly selected. The closest (in terms of distance) competitor to the offspring is chosen and competes to the

offspring to decide whether the offspring should be kept in the population. The author claimed that this form of tournament should restrict an entering element from competing with others too different from it.

Filipović *et al.* [48] investigated a fine-grained tournament selection method for a simple plant location problem in GAs. They argued that standard tournament selection does not allow precise setting of the balance between exploration and exploitation [18]. In their fine grained tournament selection method, the tournament size is not fixed but close to a pre-set value. They claimed that the fine grained tournament selection makes the ratio between exploration and exploitation able to be set precisely, and that the method solves the simple plant location problem successfully.

Luke and Panait [112] developed two modified tournament selection methods in GP. The methods use *buckets* to apply lexicographic parsimony pressure on program selection for problem domains where few individuals have the same fitness. Each individual in the bucket is treated as if it had the same fitness as others in the same bucket. They concluded that the methods maintain the same mean best-fitness-of-run as the Koza-style depth limiting, but produce equivalent or significantly lower mean tree sizes. They also developed *double tournament* and *proportional tournament* methods in GP [111]. In double tournament, individuals must pass two layers of tournaments (one by size, one by fitness) to be selected. In proportional tournament, the tournament sometimes picks winners by size, and sometimes by fitness, determined by a probability. The two variations were tested on artificial ant, 11-bit Boolean multiplexer, symbolic regression, and even-5 parity problems and compared with the depth-limiting bloat control method. They concluded that the two variations by themselves lowered total tree size only slightly in comparison to the depth-limiting bloat control method, but when combined with depth limiting the two variations yielded tree sizes at half the normal size without affecting the effectiveness of the GP system.

Matsui [119] developed two variations of tournament selection to improve the population diversity in GA. One variation is called *correlative tournament selection* which is used to select the second parent for crossover. After choosing the first parent based on the fitness value through standard tournament selection,

the Hamming distance to the first parent is used as the selection criteria for the second parent. The larger the Hamming distance, the higher the selection probability. The other is called *correlative family-based selection* which is used to choose two individuals amongst two parents and two offspring — the family — into the next generation. After producing two offspring, an individual with the highest fitness value in the family is selected for survival. The other surviving individual has the largest Hamming distance to the first surviving individual amongst the remaining three family members. The author tested the methods on the Royal Road and the non-stationary knapsack problems and concluded that the methods could improve the search performance and the genotype diversity in GA.

Poladian [142] argued that building blocks in the best individuals are likely to be disrupted by crossover and the worst individuals are unlikely to have valuable building blocks to contribute in the context of GA. In order to preserve building blocks, the author explicitly excluded the best and the worst individuals in tournament and selected the middle-ranked ones for crossover. The method was tested using tournament size 4 in both generational and steady-state GA on the hierarchical if and only if function [191] and the one-dimensional Ising spin-glass model with random coupling coefficients [153]. The method was compared with three methods that selected two parents completely random, ranked at the top in the tournament, and ranked at the bottom in the tournament, respectively. The experimental results demonstrated the benefits of the method.

At approximately the same time as our research was being conducted a related work, unbiased tournament selection, was published by Sokolov and Whitley [173]. The authors believed that a bias present in standard tournament selection is the potential for better individuals not to be selected for recombination. Therefore, they developed their unbiased tournament selection that "lines up two different permutations of the population and performs a pairwise comparison" with a constraint, which forces compared individuals to be distinct. Consequently, their method can ensure that every individual is sampled at least once. Tournament size 2 was used to test the unbiased tournament selection on three problems, one with permutation-based solution representation and two under bit encodings. Although the advantage of a generational genetic algorithm using the

unbiased tournament selection varied for different population sizes on the three problems, the authors concluded that the impact of the bias is significant, and the unbiased tournament selection provides better performance than other selection methods, including standard tournament selection, a rank based selection and fitness proportionate selection.

The literature reveals that many alternative tournament selection methods have been developed since the 1990s. However, their effectiveness is mainly demonstrated through experiments. The lack of formal models and analyses makes it difficult to understand the behaviour of these different tournament selection strategies, and to extend and develop new strategies.

2.4.2 Other parent selection methods

2.4.2.1 fitness proportionate selection

Fitness-proportional selection method selects programs according to their relative fitness values. It was introduced by Holland [74]. Koza used fitness-proportionate selection through his book [93]. In a population *P* with *N* programs, each program *i* is given a probability of being selected:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{2.8}$$

where f_i is the fitness value of the *i*th program.

One implementation of the fitness proportionate selection method is *roulette wheel selection*. A roulette wheel is divided into *N* partitions s_1, s_2, ..., s_N, where each partition s_i has a size proportional to f_i. When there is a need to select a program, the roulette wheel is turned. If the ball stops in partition s_i, program i is selected.

Although this selection method has been widely used, there are several drawbacks. If differences between high fit and low fit programs are large, the high fit programs will dominate the process of producing offspring, thus reducing the population diversity [19]. Furthermore, it is hard to control the selection pressure in the fitness-proportional selection method.

2.4.2.2 ranking selection

Ranking selection was introduced in [58]. It was designed to reduce the side effect to the population diversity in fitness-proportional selection. The method sorts programs based on their fitness values. The rank N is assigned to the best program and the rank 1 to the worst. It then uses a function (either linear [15], exponential [19], or polynomial [69]) to calculate the selection probability based on their ranks. An example of a linear function is shown in Equation 2.9:

$$p_i = \frac{1}{N}\left(1 - \eta + (2\eta - 2)\frac{i-1}{N-1}\right) \tag{2.9}$$

where, i is the rank of a program and N is the population size, η controls the selection bias to adjust the selection pressure and should meet conditions $1 \leq \eta \leq 2$. When $\eta = 1$, it gives no selection pressure as the probability is uniform ($\frac{1}{N}$). When $\eta = 2$, it gives the highest selection pressure. The probability of the best program to be selected is $\frac{\eta}{N}$.

An example of an exponential function is shown in Equation 2.10:

$$p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}} \tag{2.10}$$

where, i is the rank of a program and N is the population size. c is the selection bias to adjust the selection pressure and $0 < c < 1$. The sum $\sum_{j=1}^{N} c^{N-j}$ normalises p_i to ensure $\sum_{i=1}^{N} p_i = 1$.

There are also certain drawbacks in this selection method. Firstly, it needs to sort all programs according to their fitness values. For a very big population, this can be time-consuming. Secondly, it exaggerates the differences amongst programs with similar fitness values so that slightly better programs can be selected more often than other similar ones [194].

2.4.2.3 fitness uniform selection

Fitness uniform selection was introduced in [78]. It was designed to preserve genetic diversity in the steady-state based EAs. Although it is necessary to have selection pressure towards fitter individuals for optimisation problems, the true

optimisation goal is usually to collect not a large number of fit individuals but a single fittest individual. Therefore, with the interest in a single individual of maximal fitness instead of a population converging to maximal fitness, the fitness uniform selection method generates a random number between the lowest and highest fitness values of the current population, then selects an individual whose fitness value is the nearest to the random number. In this way, if the initial fitness distribution is not uniform, for instance only a couple of individuals with better fitness and many individuals with worse fitness or vice versa, then individuals of *low populated* fitness levels are effectively favoured regardless of whether the individual's fitness is better or worse until the population becomes fitness uniform. Therefore, with the fitness uniform selection takeover [57] never happens; the searching may waste time on the wrong end of the fitness scale, which may be beneficial to certain problems.

2.4.2.4 reserve selection

Reserve selection was introduced in [29]. It was designed to preserve possible building blocks hosted in less-fit individuals in order to prevent premature convergence.

When generating the next generation, offspring are divided into two parts with predefined sizes. Offspring that are generated through normal fitness-based selection, crossover and mutation form the first part of the population called "non-reserved area". Parents that are used to generate offspring in the non-reserved area are marked and are not again used as parents to generate offspring for the other part of the population. The other part is called "reserved-area". Offspring in this area are generated by selecting parents based on a measure called "uniqueness" instead of fitness. To calculate the uniqueness of a given parent, reserve selection sorts the current population based on fitness first. It then assigns the absolute difference of the fitness values of two immediate surrounding individuals of the given parent as the uniqueness to the given parent.

This selection method has been tested in GA on several global optimisation problem domains [29], including multimodal function optimisation, travelling

salesman problem, and multiple sequence alignment, and certain deceptive[8] problems [30], including an order-3 deceptive problem and a highly deceptive 2D problem. The experimental results demonstrated the effectiveness and robustness of the reserve selection in suppressing premature convergence and solving deceptive problems, and an enhancement in global optimisation capacity.

2.4.2.5 truncation selection

Truncation selection is most often used in GAs. In truncation selection a population is ordered by fitness, and a proportion t (e.g. $t = 1/2$ or $1/3$) of the fittest individuals are selected and reproduced $1/t$ times.

2.4.2.6 others

The literature includes many other parent selection methods not completely categorised in the above selection schemes. Some interesting examples are listed below:

- Law and Szeto [102] developed two methods to select parents for crossover based on Hamming distance in GAs.
- Smorodkina and Tauritz [172] and Holdener and Tauritz [73] introduced a couple of mate selection methods which remove user-defined parameters at the parent selection stage, by allowing individuals to self-organise into pairs of mates.
- Chellapilla [28] used an EP-style tournament selection [50] with ten opponents to select parents for the next generation.

2.5 Fitness Evaluation Cost

Fitness-driven selection methods require fitness values to be calculated in advance. Fitness evaluation is almost always the most time-consuming operation

[8]Deception, in general, refers to solutions that lead the search toward poor local optima. Deception can occur when very different solutions exist with the same fitness but their recombination leads to poor fitness. It can also occur when solutions that have relatively good fitness are not amenable to further improvement [60].

in EAs [56, 218]. It is directly connected to the efficiency of the parent selection phase; the smaller the number of fitness evaluations, the more efficient the parent selection process. Fitness evaluation cost remains an important selection-related open issue.

2.5.1 Studies in GAs

Sastry *et al.* [161] introduced the notion of fitness inheritance and showed some very promising results in reducing the number of evaluations for the OneMax problem when the population size is fixed. Kim and Cho [91] used k-means to cluster the whole population and used Euclidean distance to estimate the fitness values of other cluster members from the fitness value based on the cluster representative for saving the fitness evaluation cost. Their method was tested on the Griewangk function, the De Jong functions, the Rastrigin function and the Schwefel function. Jin and Sendhoff [86] also used k-means to cluster the whole population. Only the chromosome closest to the cluster centre is evaluated. Fitness values of other chromosomes are estimated by a neural network ensemble. Their approach was tested on the Ackley function, the Rosenbrock function, and the Sphere function.

2.5.2 Studies in GP

Altenberg [4] and Tackett [178] used a small fraction of training fitness cases to evaluate a large number of offspring produced by their brood recombination crossover operator. Giacobini *et al.* [56] used a statistical method to select a fraction of all fitness cases for evaluating programs in order to reduce the computational cost. They concluded that once the number of fitness cases is greater than a threshold, a reliable and stable convergence behaviour can be observed in their Boolean function and discrete step function problems.

Jackson [84] introduced a fitness evaluation avoidance method to avoid evaluating offspring generated by so-called fitness-preserving crossover. In his method, all nodes in a program are initially marked as *not-visited*. When a fitness case is fed to a fitness function and causes a node of the program to be evaluated, the

node is then marked as *visited*. If a program P_1 is selected for crossover and the root of a sub-tree from another program P_2 replaces a not-visited node of P_1, then the generated child cannot act differently from its parent P_1, because the inserted subtree will never be executed. Therefore, there is no need to re-evaluate the fitness of the offspring. The effectiveness of the method depends on the fraction of nodes in the programs that are not evaluated for any of the fitness cases. For the Boolean function set that Jackson used, this fraction is high; for function sets without *if* or short-circuited Boolean operators, the fraction would be low, and other techniques for saving fitness evaluation would be needed.

Wong and Zhang [199] introduced a subtree caching using a hashing for equivalence method, which caches program subtrees while taking into account algebraic equivalences between these programs, to reduce the fitness evaluation cost. The researchers tested the method on two symbolic regression problems and four classification problems and concluded that the method could provide a significant reduction in the number of node evaluations and CPU time without deteriorating the effectiveness of the system.

Luke *et al.* [110] proposed a shrinking strategy using a *diagonal layout* to gradually decrease the population size towards zero during a GP run. The method employs a large population at the beginning, then reduces the size linearly at each generation. They concluded that "decreasing the population size is always as good as, and frequently better than, various fixed-sized population strategies".

Fernandez *et al.* [46] developed a method for solving the code bloat problem[9] by taking advantage of the dynamic population. The method removes some individuals at every generation and compensates for the increase in the size of other individuals. They claimed that the method can save computing time while looking for solutions.

Rochat *et al.* [156] introduced a combination of two techniques, *island model* [179] and *plague* [187], to dynamically change the population size at run time to reduce the fitness evaluation cost.

[9]Code bloat refers to a continuous, uncontrollable increase in the size of individuals using a variable-length representation, including neural networks, finite state automata, and rule sets [14].

2.6 Overview of Genetic Operators

There are three commonly-used genetic operators: reproduction, mutation, and crossover. When to apply these operators is controlled by corresponding probability settings. In some conventional tree-based GP systems, the sum of the reproduction, crossover, and mutation probabilities is 100%. Some variants of GP systems do not follow the convention for their own special purposes. For example, the crossover and mutation operators are independent of each other so that the mutation operator is applied regardless of whether a program has also been selected for crossover [52]. These probabilities could also be updated dynamically in order to impose a constant parsimony pressure on competing tree-schemata regardless of the complexity of evolved structures [158].

2.6.1 Reproduction

The reproduction operator is the basic engine of Darwinian natural selection and survival of the fittest [93]. For reproduction, a program is selected from the current population and inserted directly without any modification into the next generation. *Elitism* [149] is a special reproduction operator. In general, elitism passes one or more of the best programs of a generation unchanged to the next generation to prevent evolution from losing the best individuals, whilst programs copied to the next generation via the reproduction operator are not necessarily the best programs of the current generation.

2.6.2 Mutation

Mutation is asexual and was categorised as the secondary genetic operator for modifying program structures in [93]. For mutation, only one parent program is selected from the population. Standard GP mutation selects a node (also called mutation point) in a parent program tree, except the root of the tree, randomly. The subtree rooted by the mutation point is replaced by a newly-generated subtree. The new program is then inserted into the next population. There are many different forms of mutation operators. Some commonly-used ones include *point* mutation, *shrink* mutation, and *hoist* or *promotion* mutation.

Point mutation [144] exchanges a randomly-chosen single node in a parent program with a random node of the same arity to ensure the new program is syntactically valid and to follow GP schema theory. Shrink mutation [8] replaces a randomly-chosen subtree in a parent program with a randomly-created terminal so that the size of the new program is smaller than its parent. Hoist or promotion mutation [92, 163] creates a new program which is a copy of a randomly-chosen subtree of a parent program. Thus the new program is also smaller than its parent and may have a different root node. Topchy and Punch [186] and Smart and Zhang [166] integrated the gradient local search technique to optimise numeric leaf values in tree-based GP.

A long list of mutation operators in GP can be found in [147].

2.6.3 Crossover

Crossover (sexual recombination) was categorised as the primary genetic operator for modifying program structures in [93] and it became the convention in almost all GP related research after that. Generally, after two programs are selected from the population, standard crossover randomly selects a node in each program tree except the root of the tree. It then exchanges the two subtrees rooted by the selected nodes (also called crossover points) between the two parent program trees to generate two new programs.

This blind replacement — randomly-chosen crossover points and ignoring the semantics of the parent programs — can often disrupt beneficial building-blocks in tree structures. In order to overcome this problem, much research has been done on understanding and improving the standard crossover operator. Ways to improve crossover include searching good offspring by integrating local search metaphors which is time-consuming, and adapting positions of crossover point which aims to reduce offspring search space. Furthermore, crossover is also modified for the code bloat problem in GP. The rest of the subsection describes these aspects.

2.6.3.1 integrating local search metaphors

Tackett [178] designed a *brood recombination* operator. The operator is inspired by the fact that animal species produce far more offspring than are expected to live. It *randomly* applies crossover N times to two chosen programs to produce $2N$ offspring. After evaluating all offspring, it puts the best two into the next generation and discards the rest of the offspring.

The brood recombination operator can be categorised as a partial local search operator because it looks for the best state in the available states but only looks at $2N$ possible successor states. Tackett asked whether the brood recombination operator reduces the diversity of subtrees, eliminating ones which are unfit in the current generation but might be useful at a later time. He compared a parent selection using standard tournament selection of size 6 with a random parent selection on the basis of using the same set of initial populations. The results demonstrated the advantage of the brood recombination operator. However, a difficulty with his conclusion is that the number of random crossover operations (the brood size factor) is chosen without sufficient regard to parent program sizes, so that the degree of intensive search within all possible successor states of chosen parents has not been well investigated.

Lang [97] introduced a *headless chicken crossover* (HCC) operator which is applied to a chosen program P and a newly- (also randomly-) generated program R. The operator repeatedly produces offspring from P by replacing a subtree of P with a subtree from R until it finds an offspring with better or equal fitness (problem solving quality) to P.

The headless chicken crossover operator can be categorised as a first-choice hill-climbing local search [160]. This is because it randomly looks for a state better than or equal to the current state and stops once it finds such a state rather than looking at all possible successors. According to [94], Lang's method is really a mutation (with hill-climbing) rather than crossover, since only one "parent" is chosen from the current generation.

Majeed and Ryan [115] introduced a *context-aware crossover* operator which identifies all possible contexts in one parent for a randomly-chosen subtree from the other parent, then evaluates each of them. The context that generates an off-

spring with the best fitness is used and the offspring generated is then passed into the next generation. Fitness proportionate selection and tournament selection with size 7 are used to select parents in different problems. The authors claimed that the operator improves both mean best fitness and mean average fitness.

The context-aware crossover operator can also be categorised as a partial local search operator. From authors' discussion of future work, it seems that they experienced a fast population convergence problem and their temporary solution was to permit only one offspring per crossover.

Hengproprohm and Chongstitvatana [68] developed a selective crossover in tree-based GP. The selective crossover tests the impact of each subtree in an individual on the overall fitness of the individual and determines the worst and the best subtrees. It then performs crossover by substituting the worst subtree of one parent with the best subtree of the other parent, combining the good subtrees from both parents to produce the offspring. The selective crossover was tested on a robot arm control and the artificial ant problems. The results demonstrated the effectiveness of the method but not the efficiency.

Harries and Smith [64] evaluated more offspring but only accepted new programs whose fitness values are better than or equal to their parents in a study of depth-based crossovers. Their search algorithm is a type of stochastic hill-climbing algorithm because not all possible offspring are evaluated and the fittest child is not necessarily chosen. Iba and Garis [79] introduced a recombinative guidance mechanism called *smart crossover* for GP, and showed the effectiveness of their approach through various experiments. Briefly, their approach uses *S-value* to evaluate performances of subtrees and selects the crossover points based on the performance evaluation results.

Yuen [208] developed two crossover operators called *simple selective crossover* and *dominance selective crossover* based on the selective crossover in GAs [190] and the uniform crossover in GP [145]. Mahfoud [114] illustrated the interaction between directed crossover operators and selection pressure in a context of genetic algorithms. Terrio and Heywood [182] investigated a family of directed crossover operators under a steady state selection model.

Other research has shown that the searching performance of an evolutionary algorithm can be improved when factorial design approaches are integrated into the crossover operator in GAs [27, 71, 72, 104, 216].

2.6.3.2 focusing on position of crossover point

The standard crossover operator selects nodes for crossover with an implicit bias towards the leaves of a program tree due to more nodes in that part of the tree in general [145]. Rosca and Ballard [157] showed that in standard crossover the average crossover point occurs near leaves for full trees. Soule and Foster [175] focused on minimal trees[10] and presented a model to describe the depth of an average crossover branch. They then simplified the model for a minimal binary tree and concluded that crossover branches are roughly one quarter as deep as the parent tree. They also concluded that depending upon the original structure of the parent tree, the standard crossover operator can be expected to swap a small, constant-sized branch in the case of full trees or a relatively large fraction of the entire tree in the case of minimal trees.

In order to remove the implicit bias toward leaves, Koza [93] introduced an alternative crossover point selection method which explicitly gives a weak selection frequency (i.e. 0.1) for leaf nodes. However, O'Reilly and Oppacher [138] pointed out that the method may still be biased towards leaf nodes under some circumstances, for instance when leaf nodes comprise less than 10% of the size of a program tree. In order to determine the effect of modifying the leaf selection frequency, Angeline [8] conducted a set of comprehensive experiments which tested four fixed and two self-adaptive leaf node selection frequencies on the Boolean 6-multiplexer function, the interlocking spirals problem, and the Wolfe Sunspot time series modeling problem. His results showed that 1) the optimal leaf selection frequency was problem-dependent and probably unpredictable without significant understanding of the given problem domain, and 2) simply removing the leaf selection frequency might be a prudent choice for many problems.

As summarised in [64], the standard crossover operator is also biased towards the bottom of a program tree. This bias might be a consequence of the depth

[10] Trees with the minimal number of nodes for their depths.

limiting [93], one of the popular bloat control techniques used in much literature. In order to remove this bias, Harries and Smith [64] proposed a depth-based crossover where each depth is given an equal probability of being chosen. Their method first randomly selects a depth then randomly selects a node at that chosen depth as a crossover point. Later, Ito *et al.* [81, 82, 83] also presented some depth control strategies with the purpose of preserving building blocks for fitness improvement, which could be interpreted as finding good crossover points. Their crossover point selection process is similar to [64] but instead of giving an equal probability to each depth, it assigns different selection probabilities to depths with bias towards the root of a program tree. In [81, 82], the depth selection probabilities are predefined and remain unchanged during evolution. Shortly after, Ito *et al.* realised that if the probability was not set properly, or was not suitable for a particular problem, the depth-dependent crossover might not work well. Therefore, they extended their work to introduce a self-tuning depth-dependent crossover operator [83]. In their extended work, each program tree is randomly assigned a depth selection probability in a predefined range rather than a fixed value during initialisation. The depth selection probability of a parent is inherited by its offspring and could be automatically adjusted during evolution according to their hypothesis that if a program is selected as a parent according to its fitness, it is likely that the depth selection probability of the program is desirable. According to Harries and Smith's and Ito *et al.*'s experimental results, ways of giving equal or unequal probabilities to each depth are demonstrated as both having some advantages.

O'Reilly and Oppacher [138] introduced a height-fair crossover operator. In a program tree, all possible subtree heights are recorded and one subtree height is randomly selected. Then within a group of subtrees of the chosen height, a random subtree is selected for swapping. Therefore, "the root and leaves of a subtree may be chosen with equal probability" [138].

Zhang *et al.* [214] introduced a looseness-controlled crossover operator in tree-based GP for object classification: a local hill-climbing search is used in constructing good building blocks, a weight called looseness is introduced to identify the good building blocks in individual programs, and the looseness values are used

as heuristics in choosing appropriate crossover points to preserve good building blocks. The looseness-controlled crossover operator was tested on a sequence of object classification problems. The results suggested that the looseness-controlled crossover operator outperforms HCC (headless chicken crossover), the standard crossover, and the standard crossover operator with hill climbing on all of the problems in terms of the classification accuracy. The approach takes slightly longer than the standard crossover operator, but it significantly improves the system efficiency over the HCC method.

Angeline [9] presented two self-adaptive crossover operators in GP. Each node is randomly assigned a probability of performing a crossover during initialisation. A roulette wheel selection is used to select a node for crossover according to the assigned probability. After performing crossover, the probability of every node in the offspring is updated using a predefined coefficient and a Gaussian random variable. Therefore, the two so-called self-adaptive crossover operators appear not to adjust the probabilities on the optimisation of the basis of an individual's fitness.

The one-point and uniform crossover operators [145] and a homologous crossover operator [99] can be also viewed as implicitly controlling the depth of the crossover points.

2.6.3.3 fighting code bloat

The earliest known report of bloating is perhaps in Pitt-approach rule systems [170]. Bloat then becomes a popular topic in GP because the fitness computation time is wasted and the readability of solutions is decreased when an increase in solution size does not correspond with fitness improvement [60].

One contributing factor to code bloat is the standard fitness-based parent selection methods, where parent sizes or structures are generally ignored. Code bloat has led to a large number of studies involving parsimony pressure [93, 111, 112, 148, 169, 210, 209, 211]. Recently, Poli *et al.* [149] confirmed that elitism can have a powerful effect on reducing bloat and larger elite sizes control bloat more strongly. Another contributing factor to code bloat is the behaviour of the standard crossover operator [175], resulting in some interesting attempts to develop

new crossover operators fighting code bloat.

Platel *et al.* [39] introduced a new recombination operator called the *Maximum Homologous Crossover* (MHC) for linear genetic programming. In contrast to the conventional crossover operator, the approach attempts to preserve similar structures from parents, by aligning them according to their homologies. To highlight disruptive effects of crossover operators, the researchers used the Royal Road Landscape and tested the homology of the new crossover operator on this landscape. Results showed a reduction in the bloat phenomenon and in the frequency of deleterious crossovers. The approach is in fact a dynamic programming like algorithm to align two programs, and it is likely to have multiple acceptable alignment results. An alignment result is randomly chosen if multiple alignments are present. Then a single point is also randomly chosen and codes below that point are swapped. Authors claimed that MHC preserves structural and lexical homology by computing an alignment, which minimises a metric of dissimilarity between parents. They claimed that MHC can keep safe similar regions of the parents, in order to favour a kind of "respect" property (the common features of parents are present in children). We think that since a linear GP program can be viewed as a Directed Acyclic Graph (DAG), aligning two DAGs and applying crossover at a single swapping point seems to be unusual.

Terrio and Heywood [181] developed a directed crossover for reducing bloat in tree-based GP. Crossover points in the method are the nodes whose contributions to the overall fitness of an individual program are maximal. Langdon [99] introduced a *size fair crossover* operator and a *homologous crossover* operator to preserve tree structures and the sizes of exchanged subtrees for controlling the code bloat problem. Majeed and Ryan [115] also claimed that the context-aware crossover operator could reduce bloat in most of their experiments and produce significantly smaller individuals in most cases.

Manrique *et al.* [118, 33] developed a grammar-based crossover. The authors claimed that the grammar-based crossover works with any grammar-based GP systems, prevents the generation of illegal trees, controls code bloat efficiently, and explores all nodes in the parents that can generate new legal individuals leading to sought-after solutions.

2.6.4 Crossover vs. Mutation

Crossover was suggested as the primary genetic operator for improving program structures in tree-based GP [93]. However, the relative effectiveness of crossover and mutation has been most controversial.

Gustafson [60] summarised that intuitively exchanging subtrees between two program trees during recombination, regardless of tree shape or content, would seem unlikely to preserve the semantic meaning of the exchanged subtrees. Thus, it is not too surprising that subtree crossover has been shown to perform similarly to mutation variants [10, 113, 139].

Gathercole and Ross [55] pointed out that in crossover, subtree discovery and movement takes place mostly near the leaf nodes, with nodes near the root left untouched. Therefore, diversity drops quickly to zero near the root. They claimed that GP is then unable to create fitter trees via crossover, leaving mutation as the only (but ineffective) route to discovery of fitter trees. Interestingly, Chellapilla [28] later demonstrated experimentally that mutation is effective.

However, where and why one is preferable to the other is strongly dependent on problem domains and parameter settings [113]. Crossover has remained the dominant genetic operator in deriving optimal solutions in the large number of attempts since the 1990s.

The debates indicate that crossover should do more than blindly exchange subtrees. This also implies that offspring selection is crucial, especially in recombination.

2.7 Typical Problem Domains in GP

This section briefly presents only three typical problem domains used in GP research. Other commonly-used problem domains can be found in [93, 14].

2.7.1 Boolean

Multiplexer and Parity are two representative problems in the Boolean problem domain [93]. Langdon and Poli [100] presented a detailed analysis of Boolean

program spaces.

For a n-multiplexer problem, the input consists of i address bits and 2^i data bit, and $n = i + 2^i$. The output of a multiplexer function is the Boolean value of the particular data bit that is chosen by the value of the address bits of the multiplexer. For an even-n-parity problem, the input is a string of n Boolean values, and the output is *true* if there are an even number of true's, and otherwise *false*. The most characteristic aspect of this problem is the requirement to use all inputs in an optimal solution and a random solution could lead to a score of 50% accuracy [60]. Furthermore, optimal solutions could be dense in the search space as an optimal solution generally does not require a specific order of the n inputs presented. For both problems, there are a very small number of fitness cases.

2.7.2 Symbolic regression

Symbolic regression differs from conventional linear regression, quadratic regression, exponential regression, and other conventional types of regression.

In conventional regression, for example, after being giving a set of values of various independent variable(s) and the corresponding values for the dependent variable(s), a user first needs to decide whether a suitable model is a linear regression, a quadratic regression, or an exponential regression, or whether to try to fit the data points to some other type of function. The user then needs to discover a set of numerical coefficients for the model involving the independent variable(s) that minimises errors between the values of the dependent variables(s) computed with the model and the given target values for the dependent variable(s).

But often, the real problem is to decide what type of model most appropriately fits the data, not merely computing the appropriate numerical coefficients after the model has already been chosen. Symbolic regression searches for both models and appropriate numerical coefficients that go with the models.

2.7.3 Classification

In general, classification means to assign items in a data set to a number of categories or classes. A classification problem for objects in a particular domain is

the problem of separating these objects into classes, and giving criteria for determining which particular class a particular object belongs to. In terms of the number of classes to be classified, classification problems can be categorised into binary and multi-class classification problems. That is, multi-class classification problems refer to classification problems with three or more classes of interest, in contrast to binary classification problems, which have only two classes.

GP was used to discover decision trees and grammars, and also used with a wrapper[11] for binary classification and pattern recognition [93]. It has became a general method for classification problems as shown by a large amount of successful research work [77, 164, 174, 177, 180, 195]. For multi-class classification problems, translating numeric values into class labels is not straightforward. There have been many attempts since the 1990s. Gathercole and Ross [54] decomposed a multi-class classification problem into a binary classification problem. Loveard and Ciesielski [109] and Zhang *et al.* [212] developed a static range selection method. The method partitions the numeric outputs of a GP classifier into multiple regions, where each region represents one class. Other attempts, including dynamic range selection [109], centred and slotted dynamic class boundary determination methods [215], and probability-based methods [168] are also very interesting and promising.

2.8 Chapter Summary

This chapter reviewed the fields of machine learning, evolutionary computation, especially GP and its components, and areas of research — parent and offspring selection — that closely related to work in this thesis.

The current situation of the research in parent and offspring selection in tree-based GP is summarised as:

- A variety of parent selection schemes in EAs have been developed and studied, as well as several selection pressure measurements. However, in GP, how parent selection pressure should be tuned has not been well addressed.

[11] This translates the numeric outputs of a GP classifier into class labels based on the sign of the numeric values.

Parent selection pressure control has not been fully understood and has only been implicitly studied experimentally in some pieces of work, like changing the sampling process for a tournament [173], changing population size [110, 156, 185], and applying parsimony pressure which is originally used for bloat control [147, 34].

- It is necessary to apply offspring selection and many attempts at offspring selection have been made. However, the impact of offspring selection on the overall GP search performance has not been well addressed, nor has the configuration between parent and offspring selection.

- Many algorithms for saving the fitness evaluation cost in parent selection have been developed. However, few algorithms exploit the characteristics of a particular selection scheme for saving the fitness evaluation cost.

- A variety of algorithms for reducing offspring search space have been developed. However, these algorithms often disagree with each other. For instance, when choosing good crossover points, reducing or retaining leaf nodes selection bias are both recommended; assigning equal or unequal probabilities to each depth in parent program trees both have advantages. Further investigation in this research area is clearly needed.

This thesis addresses these selection-related open issues in tree-based GP via a series of carefully-designed experiments and analyses.

Part I

Analysing Parent Selection Behaviour

Chapter 3

Tuning Parent Selection Pressure

This chapter firstly presents an analysis of the standard tournament selection with mathematical models and visualisations, showing the working of the standard tournament selection and revealing the impacts of tournament size and population size on parent selection pressure. It then addresses three issues that influence the parent selection behaviour in the standard tournament selection. Concerning the first issue, this chapter discusses the existing tournament selection but in a less commonly-used form, and concerning the other two issues, this chapter introduces two new approaches to improving the standard tournament selection scheme respectively, followed by analyses of their selection behaviours, to demonstrate how parent selection pressure should be properly controlled.

3.1 Introduction

To determine which parent selection scheme is suitable for a particular evolutionary learning paradigm, three factors need to be considered. The first factor is whether selection pressure of a selection scheme can be changed easily because it directly affects the convergence of learning. The second is whether a selection scheme supports parallel architectures because a parallel architecture is very useful for speeding up learning paradigms that are computationally intensive. The third factor is whether the time complexity of a selection scheme is low because the running cost of the selection scheme can be amplified by the number of individuals involved.

As introduced in the previous chapter, tournament selection is one of the most commonly-used parent selection schemes in EAs. Standard tournament selection randomly draws/samples *k* individuals with replacement from the current population of size N into a tournament of size k and selects the one with the best fitness from the tournament. In general, selection pressure in tournament selection can be easily changed by using different tournament sizes. Drawing individuals with replacement into a tournament makes the population remain unchanged, which in turn allows tournament selection to easily support parallel architectures. Selecting the winner involves simply ranking individuals in a tournament of size k. Furthermore, in general, since the standard breeding process in GP produces one offspring by applying mutation to one parent and produces two offspring by applying crossover to two parents, the total number of tournaments needed to generate the entire next generation is N. Therefore, the time complexity of tournament selection is $O(kN)$.

GP is recognised as a computationally-intensive learning method, requiring a parallel architecture to improve its efficiency. Furthermore, it is not uncommon to have millions of individuals in a population when solving complex problems [96], thus sorting a whole population is time consuming. The parallel architecture support and the linear time complexity have made standard tournament selection very popular in GP.

Due to the popularity of standard tournament selection in GP, this thesis focuses on tournament selection to analyse parent selection behaviour and provide guidance on how to tune the parent selection pressure properly.

According to the literature, there are some possible ways to influence the parent selection pressure in tournament selection, including changing the sampling process for a tournament [173], changing population size [110, 156, 185], and changing the tournament size. However, it is still not clear how to make changes on these elements to control the selection pressure along evolution effectively. Especially, it is not clear how to set the tournament size for different sized populations. A common opinion sensed throughout the literature is that the tournament size configuration should depend on the population size [61].

Although standard tournament selection is very popular in GP, it still has

some open questions as well as some drawbacks. For instance, because individuals are sampled with replacement, it is possible to have the same individual sampled multiple times in a tournament (referred as the *multi-sampled issue* in this thesis). It is also possible to have some individuals not sampled at all when using small tournament sizes (referred as the *not-sampled issue* in this thesis). Some researchers believe that the two issues may lower the probability of some reasonably good individuals being sampled or selected, while other researchers have an intuition that they are not important. These views have not yet been sufficiently proven. In addition, although in general the selection pressure can be changed to influence the convergence of the genetic search process by using different tournament sizes, we realised that during population convergence (i.e., groups of programs having the same or similar fitness values), the selection pressure between groups increases, resulting in "better" groups dominating the next population and possibly causing premature convergence (referred as the *high between-group selection pressure issue* in this thesis). Therefore, tournament size itself is not adequate for tuning parent selection pressure. There exists a strong demand to clarify the open issues and solve the drawbacks of standard tournament selection in order to conduct an effective selection process in GP. To do that, a thorough investigation of tournament selection is necessary.

3.2 Chapter Goals

In order to tune the parent selection pressure, including the initial static configuration and the later dynamic control or adjustment, it is necessary to first of all understand the process of tournament selection, then to understand the changes in selection process along evolution. In particular, this chapter addresses the following research questions:

1. What is the relationship between tournament size, population size and selection pressure?

2. Are the above open issues in standard tournament selection critical?

3. How should the standard tournament selection be modified in order to control the parent selection pressure properly?

Section 2.4.1 (on page 18) reviewed several selection pressure measurements in tournament selection, as well as the models of its sampling and selection behaviours. From those related studies in the last decade or so, it is clear that similar models are re-presented but used for different research purposes. This thesis extends some related models for the research goals.

3.3 Assumptions and Definitions

This section presents the assumptions and definitions necessary to model and analyse tournament selection.

In general, a population can be partitioned into bags consisting of programs with equal fitness. These "fitness bags" may have different sizes. As each fitness bag is associated with a distinct fitness rank, we can characterise a population by the number of distinct fitness ranks and the size of each corresponding fitness bag, which we term *fitness rank distribution* (FRD). If S is the population, then we use the notation N to be the size of the population, S_j to be the bag of programs with the fitness rank j and $|S_j|$ to be its size, and $|S|$ to be the number of distinct fitness bags. We denote tournament size by k and rank the program with the worst fitness 1st. We follow the standard breeding process, that is, one parent produces one offspring after mutation and two parents produce two offspring via crossover. Therefore the total number of tournaments is N at the end of generating all individuals in the next generation. Although tournaments indeed can be implemented in a parallel manner, we also assume that they are conducted sequentially so that the number of tournaments conducted reflects the progress of generating the next generation.

To analyse selection behaviour, we chose two commonly used measures, i.e., the loss of program diversity and the selection frequency (see page 20). We also developed a new measure called *selection probability distribution*.

The selection probability distribution of a population is defined as consisting of the probabilities of each individual in the population being selected at least

once in the selection phase. It is illustrated in a three dimensional graph, where the x-axis shows every individual in the population ranked by fitness, the y-axis shows the number of tournaments conducted in the selection phase (from 1 to N), and the z-axis is the selection probability of a program being selected at least once in a corresponding number of tournaments. Therefore, the measure provides a full picture of the selection behaviour over the population during the whole selection phase. Figure 3.1 shows the selection probability distribution measure for the standard tournament selection of tournament size 4 on a wholly diverse population of size 40.

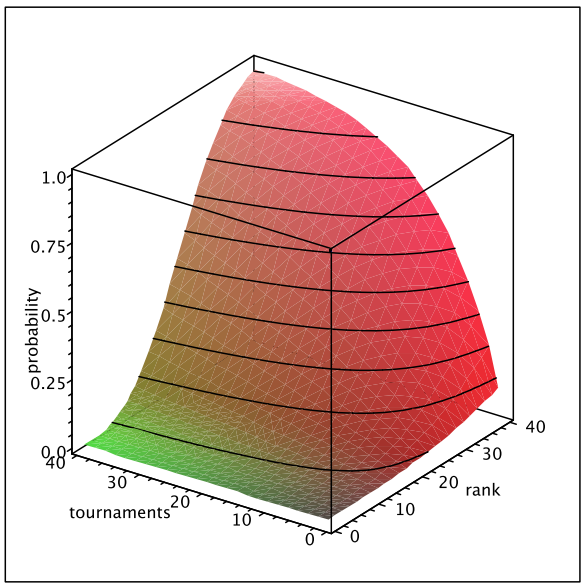


Figure 3.1: An example of the selection probability distribution measure.

We use these three measures on four populations with four different fitness rank distributions, namely *uniform*, *reversed quadratic*, *random*, and *quadratic* fitness rank distributions. The four fitness rank distributions are designed to simulate the four stages of evolution. The uniform fitness rank distribution represents the initialisation stage, where each fitness bag has a roughly equal number of programs. A typical case of the uniform fitness rank distribution can be found in a wholly diverse population. The reversed quadratic fitness rank distribution represents the early evolving stage, where commonly only few individuals have better fitness values. The random fitness rank distribution represents the middle stage of evolution, where better and worse individuals are possibly randomly distributed. The quadratic fitness rank distribution represents the later stage of

evolution, where a large number of individuals have converged to better fitness values.

Since the impact of population size on selection behaviour is unclear, we tested several different commonly-used population sizes, ranging from small to large. This chapter illustrates only the results for three population sizes, namely 40, 400, and 2000, for the uniform FRD, the random FRD, and the reversed quadratic and quadratic FRDs respectively. Note that although the populations with different FRDs are of different sizes, the number of distinct fitness ranks is designed to be the same value (i.e. 40) for easy visualisation and comparison purposes (see Figure 3.2). We also studied and visualised other different numbers of distinct fitness ranks, including 100, 500 and 1000, and obtained consistent results.

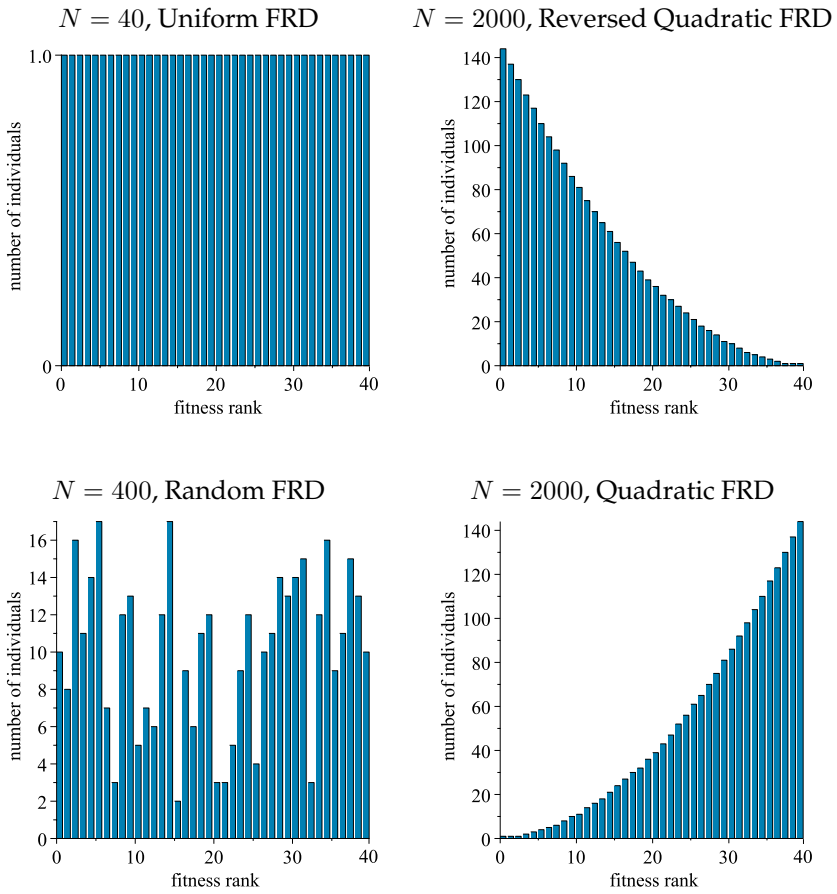


Figure 3.2: Four populations with different fitness rank distributions.

3.4 Analysis of Relationship

This section analyses the relationship between tournament size, population size, and selection pressure in standard tournament selection. It starts with models of the sampling and selection behaviours of standard tournament selection, followed by the analyses of the loss of program diversity, the selection frequency, and the selection probability distribution.

3.4.1 Sampling probability modelling

For any program p, let I_y be the event that p is drawn or sampled at least once in $y \in \{1, ..., N\}$ tournaments. As sampling is independent of the program rank, the FRD of a population does not have any impact on the sampling behaviour. Since an individual can be sampled multiple times in a single tournament provided the tournament size is greater than one, the probability of I_y is:

$$\begin{aligned} P(I_y) &= 1 - \left(\frac{N-1}{N}\right)^{yk} \\ &= 1 - \left[\left(\frac{N-1}{N}\right)^{N}\right]^{\frac{y}{N}k} \end{aligned} \tag{3.1}$$

According to Equation 3.1, we can simulate the probability trends of a single program being sampled at least once in the standard tournament selection. Figure 3.3 illustrates the sampling probability trends using six different tournament sizes (1, 2, 4, 7, 20 and 40) in three populations with different sizes (40, 400, and 2000) in the selection phase, as the number of tournaments increases up to the corresponding population size.

The figure shows that the larger the tournament size, the higher the sampling probability. Furthermore, the probability of an arbitrary program being sampled increases with increasing numbers of tournaments. For a given tournament size, the sampling probability of a program after a given number of tournaments decreases with increasing population size.

However, interestingly, for a given tournament size, the trends of sampling probabilities of a program in the selection phase (along the increments of the

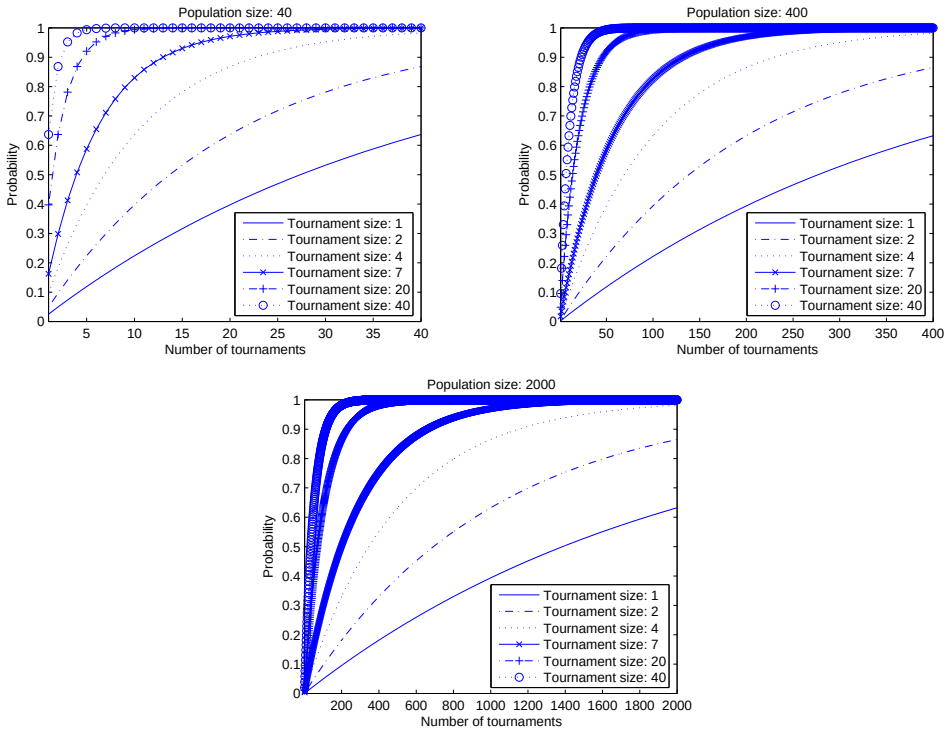


Figure 3.3: Trends of the probability that a program is sampled at least once in the standard tournament selection in the parent selection phase. (Note that the scales on the x-axes differ.)

number of tournaments) are very similar in different-sized populations. This is because $(\frac{N-1}{N})^N$ is close to a constant e^{-1} for large N, so sampling probability depends on $\frac{y}{N}$ when k is fixed. In other words, it depends on the fraction of population generated for the next generation when the tournament size is fixed. Therefore, with the same tournament size, sampling probability for large populations at a stage of generating a given fraction of the population can be estimated reliably from experiments on smaller populations at the stage of generating the same fraction of population.

3.4.2 Selection probability modelling

In general, within a population, some programs share the same fitness value. A wholly diverse population in which every individual has distinct fitness value is an uncommon simple situation. This special simple situation may occur when constructing the initial generation with constraints explicitly applied, and when the population size is less than the number of possible unique fitness values. Most

papers in the literature study only the standard tournament selection in a simple wholly diverse population. In contrast, this thesis focuses on a more general situation where some programs have the same fitness value and therefore have the same rank.

Modelling the selection probability of a program ranked jth in a general situation is difficult because the probability will be affected by the number of programs with the same rank, the probability of any of these programs being sampled, and the probabilities of any programs with worse fitness values being sampled. The rest of this subsection presents the selection probability model, followed by a proof.

Lemma 1. *If $E_{j,y}$ is the event that $p \in S_j$ is selected at least once in $y \in \{1, ..., N\}$ tournaments, the probability of $E_{j,y}$ is:*

$$P(E_{j,y}) = 1 - \left(1 - \frac{1}{|S_j|}\left(\left(\frac{\sum_{i=1}^{j}|S_i|}{N}\right)^k - \left(\frac{\sum_{i=1}^{j-1}|S_i|}{N}\right)^k\right)\right)^y \tag{3.2}$$

Proof. The probability that all the programs sampled for a tournament have a fitness rank between 1st and jth (i.e. are from $S_1, \ldots, S_j$) is given by

$$\left(\frac{\sum_{i=1}^{j}|S_i|}{N}\right)^k$$

If T_j is the event that the best-ranked program in a tournament is from S_j, the probability of T_j (i.e, the selected program will have rank j) is:

$$P(T_j) = \left(\frac{\sum_{i=1}^{j}|S_i|}{N}\right)^k - \left(\frac{\sum_{i=1}^{j-1}|S_i|}{N}\right)^k \tag{3.3}$$

Let W_j be the event that the program $p \in S_j$ wins or is selected in a tournament. As each element of S_j has equal probability of being selected in a tournament, the probability of W_j is:

$$P(W_j) = \frac{\left(\frac{\sum_{i=1}^{j}|S_i|}{N}\right)^k - \left(\frac{\sum_{i=1}^{j-1}|S_i|}{N}\right)^k}{|S_j|} \tag{3.4}$$

Therefore the probability that $p \in S_j$ is selected at least once in y tournaments is

$$P(E_{j,y}) = 1 - (1 - P(W_j))^y$$

Replacing $P(W_j)$, we obtain Equation (3.2). □

For the special simple situation that all individuals have distinct fitness values, $|S_j|$ becomes 1 and Equation (3.4) reduces to

$$P(W_j) = \frac{j^k - (j-1)^k}{N^k} \tag{3.5}$$

which is identical or equivalent to models presented in [15, 127].

3.4.3 Loss of program diversity analysis

We calculate the total loss of program diversity using Equation 2.3 (on page 20) in which $P(W_j)$ is replaced by Equation 3.4. We also split the total loss of program diversity into two parts. One part is from the fraction of the population that is not sampled at all during the selection phase. We calculate it also using Equation 2.3 by replacing $1 - P(W_j)$ with $\left(\frac{N-1}{N}\right)^k$, which is the probability that an individual has not been sampled in a tournament of size k. The other part is from the fraction of population that is sampled but never wins any tournament (i.e., not selected). We calculate it by taking the difference between the total loss of program diversity and the contribution from not-sampled individuals.

Figure 3.4 shows the three loss of program diversity measures, namely the *total* loss of program diversity and the contributions from *not-sampled* and *not-selected* individuals for the standard tournament selection.

Figure 3.4 shows that when the tournament size is 1, the total loss of program diversity is entirely due to the not-sampled individuals. This is because once an individual is sampled, it must be selected as a parent as there are no other competitors in the tournament. However, the contribution from not-sampled individuals reduces to zero as the tournament size increases. For instance, the contribution from not-sampled individuals is 13.5%, 5.0%, and 1.8% for tournament sizes 2, 3, and 4 respectively for all different populations. On the other hand, the

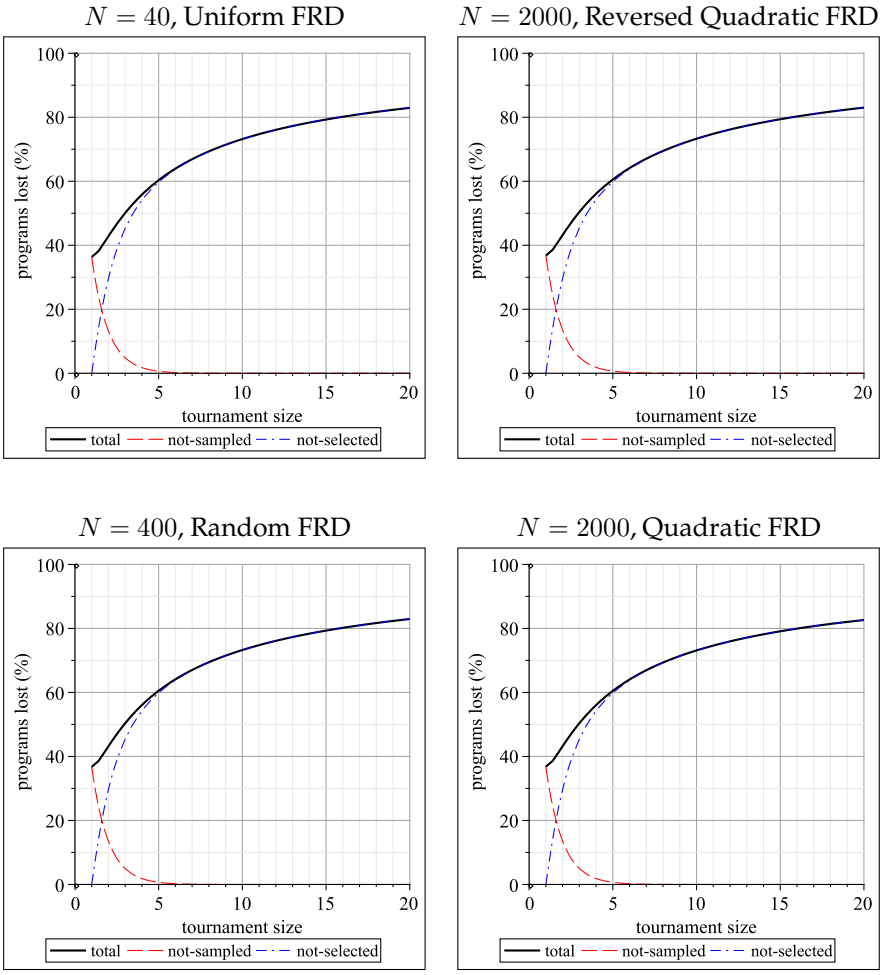


Figure 3.4: Loss of program diversity in the standard tournament selection scheme on four populations with different FRDs. Note that the tournament size is discrete but the plots show curves to aid interpretation.

contribution from not-selected individuals becomes larger and completely dominates the total loss of program diversity when the tournament size is greater than five.

We found that overall there were no noticeable differences between the three loss of program diversity measures on the four different populations with different FRDs. The loss of program diversity measure in the standard tournament selection depends almost entirely on the tournament size, and is almost independent of the FRD. Therefore, the loss of program diversity measure cannot capture the effect of different FRDs. Later, we show that FRD is significant to selection behaviour, implying that the loss of program diversity is not an adequate measure.

Extra visualisations on other-sized populations and different numbers of distinct fitness values with the four FRDs support the finding. While it is difficult to

prove the finding mathematically, the following brief analysis of the contribution from not-sampled individuals may help explain the finding.

For the standard tournament selection, according to Equation 3.1, the probability that a program has never been sampled in $y = N$ tournaments is:

$$\left(\left(\frac{N-1}{N}\right)^N\right)^{\frac{N}{N}k} = \left(\frac{N-1}{N}\right)^{Nk} \approx e^{-k} \tag{3.6}$$

for large N. The loss of program diversity contributed by not-sampled individuals is approximately:

$$\frac{1}{N}\sum_{i=1}^{N} e^{-k} = e^{-k} \tag{3.7}$$

which is just a function of the tournament size k. Therefore, the trends of the loss of program diversity contributed by not-sampled individuals are almost the same in the four different-sized populations with different fitness rank distributions. For the total loss of program diversity, we may obtain a function of a similar form after simplifying or approximating Equation 3.2 on page 53.

It is clear that for the standard tournament selection scheme, tournament size ranging from 1 to 5 is in fact a double-edged sword. Increasing the sampling probability using larger tournament sizes will decrease the selection probability. On the other hand, increasing the selection probability using smaller tournament sizes will decrease the sampling probability.

Now we have the following two questions from the analyses:

- How can we modify the standard tournament selection scheme to minimise total loss of program diversity by reducing the loss of program diversity contributed by not-sampled individuals without increasing the tournament size?

- Does the reduced loss of program diversity in such a new selection scheme significantly improve the GP search performance?

These questions will be addressed in Section 3.6 in this chapter.

3.4.4 Selection frequency analysis

For each of the four populations with different FRDs, we calculate the expected selection frequency of a program in the selection phase based on Equation 2.4 (on page 20) and our probability model of a program being selected in a tournament (Equation 3.4), that is $N \times P(W_j)$.

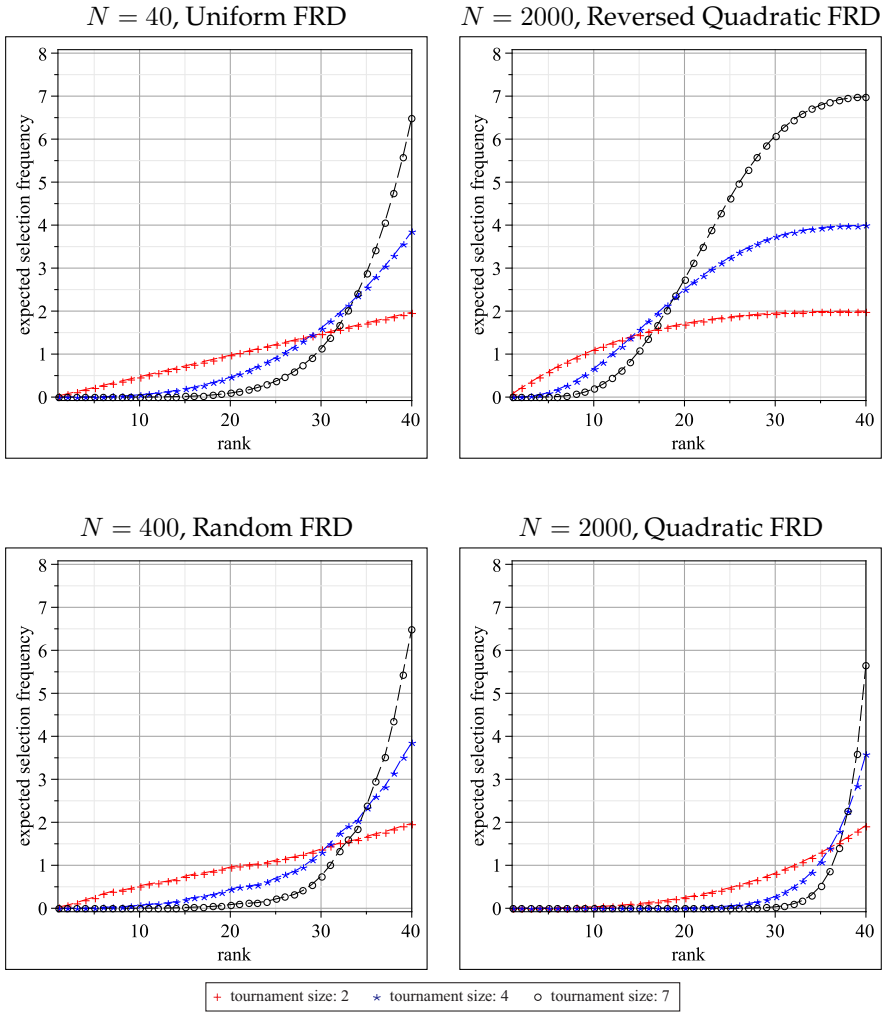


Figure 3.5: Selection frequency in the standard tournament selection scheme on four populations with different FRDs.

Instead of plotting the expected selection frequency for every individual, we plot it only for an individual in each of the 40 unique fitness ranks so that plots in different-sized populations have the same scale and it is easy to identify what fitness ranks may be lost. Furthermore, we chose three different tournament sizes (2, 4, and 7) commonly used in the literature, to illustrate how tournament size affects the selection behaviour, as shown in Figure 3.5. Note that extra visualisa-

tions on other-sized populations with the given four FRDs are consistent with the figure.

The figure shows that overall the standard tournament selection favours better-ranked individuals for all tournament sizes: the expected selection frequencies for better individuals are higher than for worse individuals. The selection pressure is biased in favour of better individuals as the tournament size increases.

The figure also shows that skewed FRDs aggravate selection bias quite significantly. For the reversed quadratic FRD, there are more individuals of worse-ranked fitness received the selection preference. The GP search will still wander around without paying sufficient attention to the small number of outstanding individuals. Ideally, in this situation, a good selection schema should focus on the small number of good individuals to speed up evolution.

For the quadratic FRD, the selection frequencies are strongly biased towards individuals with better ranks. The population diversity will be quickly lost, the convergence may speed up, and the GP search may be confined in local optima. Ideally, in this situation, a good selection scheme should slow down the convergence.

Interestingly, by comparing the results of the selection frequency measure of the uniform FRD and the random FRD, we expected to see some differences but there were not and the shapes were very similar. This may imply that the standard tournament selection may tolerate the difference between the uniform and random FRDs, and therefore sometimes take long time to converge. To interpret this finding, we offer the following analysis.

Let μ be the average number of individuals in each S_j. In the uniform FRD, for all $j \in \{1, ..., |S|\}$, $|S_j| = \mu$. While in the random FRD, it has

$$\frac{\sum_{i=1}^{j} |S_i|}{j} \approx \mu \tag{3.8}$$

and the approximation becomes more precise when j is close to $|S|$. As the selection frequency for a program p of rank j is $N \times P(W_j)$, we simplify $P(W_j)$ for the

uniform FRD as:

$$P(W_j) = \frac{\left(\frac{j\mu}{|S|\mu}\right)^k - \left(\frac{(j-1)\mu}{|S|\mu}\right)^k}{\mu} = \frac{1}{\mu|S|^k}\left(j^k - (j-1)^k\right) \tag{3.9}$$

and for the random FRD as:

$$P(W_j) \approx \frac{\left(\frac{j\mu}{|S|\mu}\right)^k - \left(\frac{(j-1)\mu}{|S|\mu}\right)^k}{|S_j|} = \frac{1}{|S_j||S|^k}\left(j^k - (j-1)^k\right) \tag{3.10}$$

From Equation 3.9, in the uniform FRD, the selection frequency for an individual of rank j will be just $\frac{1}{|S|^{k-1}}\left(j^k - (j-1)^k\right)$, which is independent of the actual number of individuals of the same rank.

From Equation 3.10, the selection frequency of an individual of rank j in the random FRD is approximately:

$$\frac{1}{|S_j||S|^k}\left(j^k - (j-1)^k\right) \times |S|\mu = \frac{\mu}{|S_j|} \times \frac{1}{|S|^{k-1}}\left(j^k - (j-1)^k\right) \tag{3.11}$$

which differs from that in the uniform FRD by a factor of $\frac{\mu}{|S_j|}$. For a random FRD, $\frac{\mu}{|S_j|}$ could be small. Therefore, only slight fluctuations and differences can be found in the figure of the random FRD under very close inspection while comparing with that of the uniform FRD. Ideally, in this situation, a good selection scheme should be able to adjust the selection pressure distinguishably according to the changes in the fitness rank distribution. For instance, it should give a relatively high selection preference to an individual in a fitness bag with a smaller size in order to increase the chance of propagating this genetic material and a relatively low selection preference to an individual in another fitness bag with a larger size in order to reduce the chance of the same.

3.4.5 Selection probability distribution analysis

For each of the four populations with different FRDs, we calculate the selection probability distribution in the selection phase based on Equation 3.2.

Figure 3.6 illustrates the selection probability distribution using the three different tournament sizes (2, 4, and 7) on the four populations with different FRDs. Again, we plot it for each of the 40 unique individual ranks.

Clearly, different tournament sizes have a different impact on the selection pressure. The larger the tournament size, the higher the selection pressure on individuals of better ranks.

For the same tournament size, we observe that same population size but different FRDs (i.e. the second and the fourth rows in Figure 3.6) may result in different selection probability distributions, indicating that the parent selection pressure is also affected by the FRD.

From additional visualisations on other-sized populations with the four FRDs, we observed that similar FRD but different population sizes result in the similar selection probability distributions, indicating that population size does not significantly influence the selection pressure. Note that in general the genetic material differs between populations of different sizes, and the impact of genetic material in different-sized populations on the GP search performance varies significantly. However, understanding that impact is another research topic and is beyond the scope of this thesis.

This experiment has clarified the relationship between tournament size, population size, and selection pressure in the standard tournament selection:

- Tournament size and population size both affect the sampling probability of an individual in a single tournament, but the trend is affected only by tournament size.
- Tournament size affects the selection pressure while population size has little impact unless the population size is very small.
- The FRD of a population also affects the selection pressure, but standard tournament selection is unable to reduce the impact of a given FRD in order to properly adjust the selection pressure.

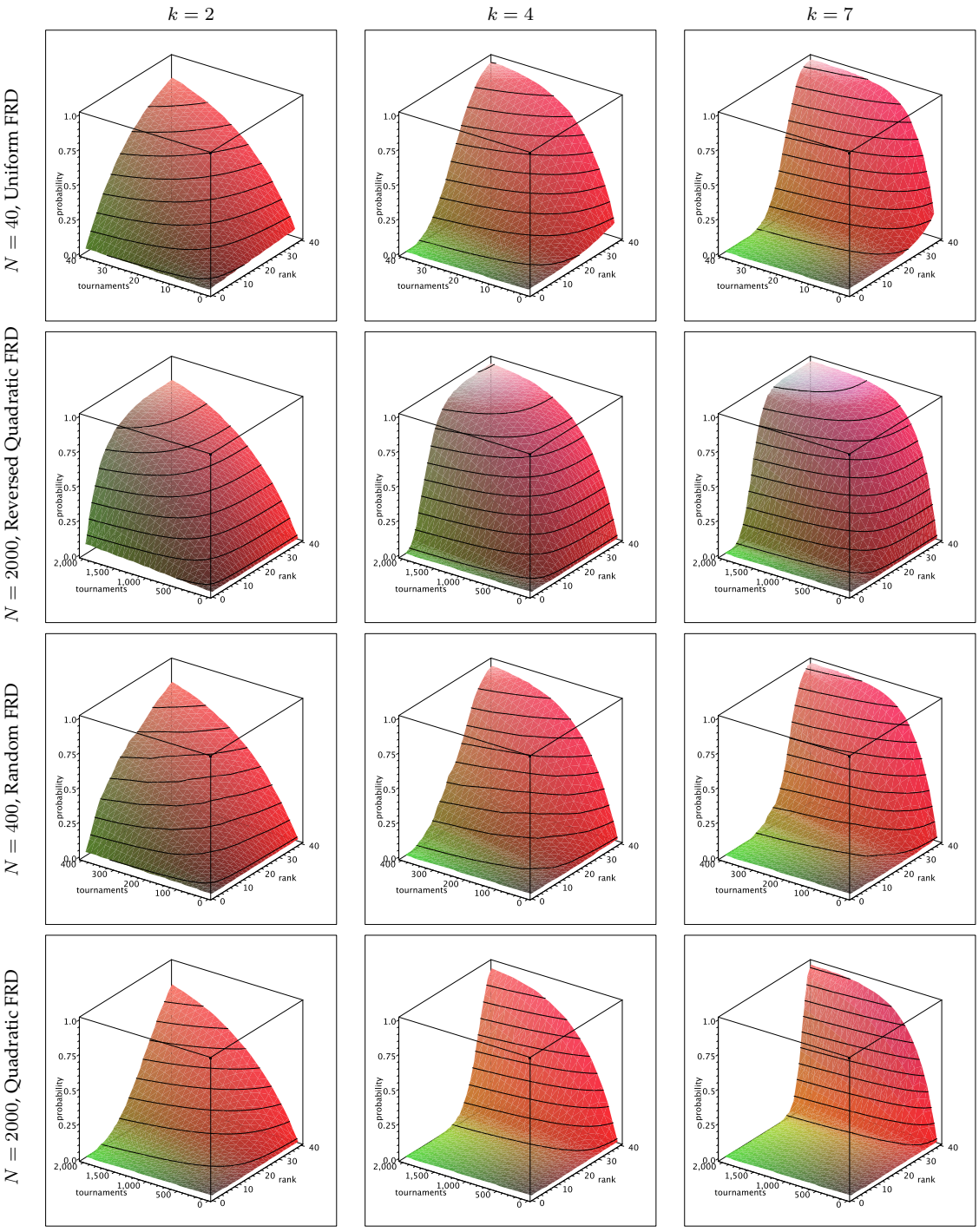


Figure 3.6: Selection probability distribution in the standard tournament selection scheme with tournament size 2, 4 and 7 on four populations with different FRDs.

The next three sections will analyse the three open issues (from page 47) in detail in order to determine how to solve them and whether they are critical to the parent selection behaviour.

3.5 Analysis of the Multi-Sampled Issue

As mentioned earlier, the impact of the multi-sampled issue is unclear. This section shows that the multi-sampled issue is not a serious problem by analysing the *no-replacement* tournament selection, which solves the multi-sampled issue. This section then compares the no-replacement tournament selection to standard tournament selection, showing there is no significant difference between them.

3.5.1 No-replacement tournament selection

According to [57], no-replacement tournament selection was introduced at the same time as standard tournament selection. It is not clear why the no-replacement tournament selection is less commonly used in EAs. The no-replacement tournament selection samples individuals into a tournament without replacement, that is, it will not return a sampled individual back to the population immediately, thus no individual can be sampled multiple times into the same tournament. After the winner is determined, it then returns all individuals of the tournament to the population.

3.5.2 Modelling no-replacement tournament selection

The only factor making no-replacement tournament selection different from the standard one is that any individual in a population will be sampled at most once in a single tournament. Therefore, if D is the event that an arbitrary program is drawn or sampled in a tournament of size k, the probability of D is:

$$P(D) = \frac{k}{N} \tag{3.12}$$

If I_y is the event that p is drawn or sampled at least once in $y \in \{1, ..., N\}$ tournaments, the probability of I_y is:

$$P(I_y) = 1 - (1 - P(D))^y = 1 - \left(1 - \frac{k}{N}\right)^y = 1 - \left[\left(\frac{N-k}{N}\right)^N\right]^{\frac{y}{N}} \tag{3.13}$$

Lemma 2. *For a particular program $p \in S_j$, if $E_{j,y}$ is the event that p is selected at least*

once in $y \in \{1, ..., N\}$ tournaments, the probability of $E_{j,y}$ is:

$$P(E_{j,y}) = 1 - \left(1 - \frac{1}{|S_j|}\left(\frac{\binom{\sum_{i=1}^{j}|S_i|}{k}}{\binom{N}{k}} - \frac{\binom{\sum_{i=1}^{j-1}|S_i|}{k}}{\binom{N}{k}}\right)\right)^{y} \tag{3.14}$$

Proof. The probability that all the programs sampled for a tournament have a fitness rank between 1 and j (i.e. are from $S_1, \ldots, S_j$) is given by

$$\frac{\binom{\sum_{i=1}^{j}|S_i|}{k}}{\binom{N}{k}}$$

If T_j is the event that the best-ranked program in a tournament is from S_j, the probability of T_j is:

$$P(T_j) = \frac{\binom{\sum_{i=1}^{j}|S_i|}{k}}{\binom{N}{k}} - \frac{\binom{\sum_{i=1}^{j-1}|S_i|}{k}}{\binom{N}{k}} \tag{3.15}$$

Let W_j be the event that the program $p \in S_j$ wins or is selected in a tournament. As each element of S_j has equal probability of being selected in a tournament, the probability of W_j is:

$$P(W_j) = \frac{P(T_j)}{|S_j|} \tag{3.16}$$

Therefore the probability that p is selected at least once in y tournaments is:

$$P(E_{j,y}) = 1 - (1 - P(W_j))^y \tag{3.17}$$

Substituting for $P(W_j)$ we obtain Equation (3.14) as required. □

For the special simple situation that all individuals have distinct fitness values, $|S_j|$ becomes 1. Substituting this into Equations (3.15) and (3.16), we obtain

the following equation, which is identical to the model presented in [22].

$$P(W_j) = \frac{\binom{j}{k} - \binom{j-1}{k}}{\binom{N}{k}} \tag{3.18}$$

3.5.3 Selection behaviour analysis

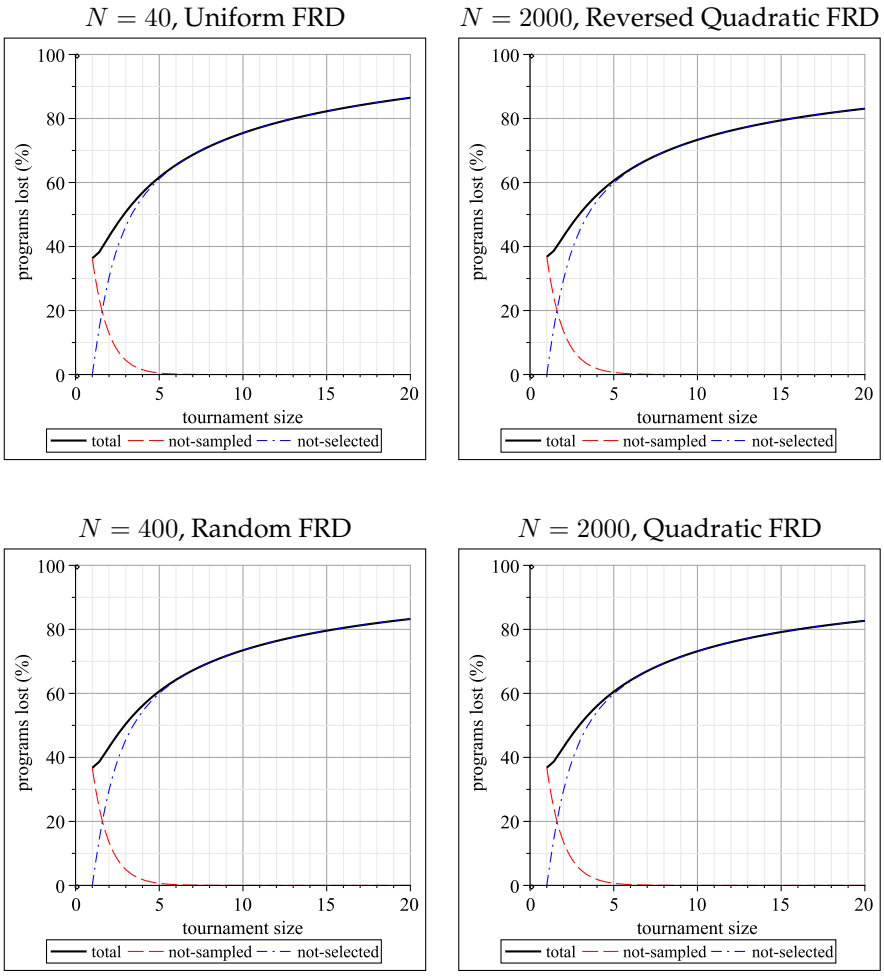


Figure 3.7: Loss of program diversity in the no-replacement tournament selection scheme on four populations with different FRDs. Note that tournament size is discrete but the plots show curves to aid interpretation.

The loss of program diversity, the selection frequency, and the selection probability distribution for the no-replacement tournament selection are illustrated in Figures 3.7, 3.8, and 3.9, respectively. Comparison results of these figures and Figures 3.4, 3.5 and 3.6 show that the selection behaviour in the no-replacement tournament selection is almost identical to that in standard tournament selection.

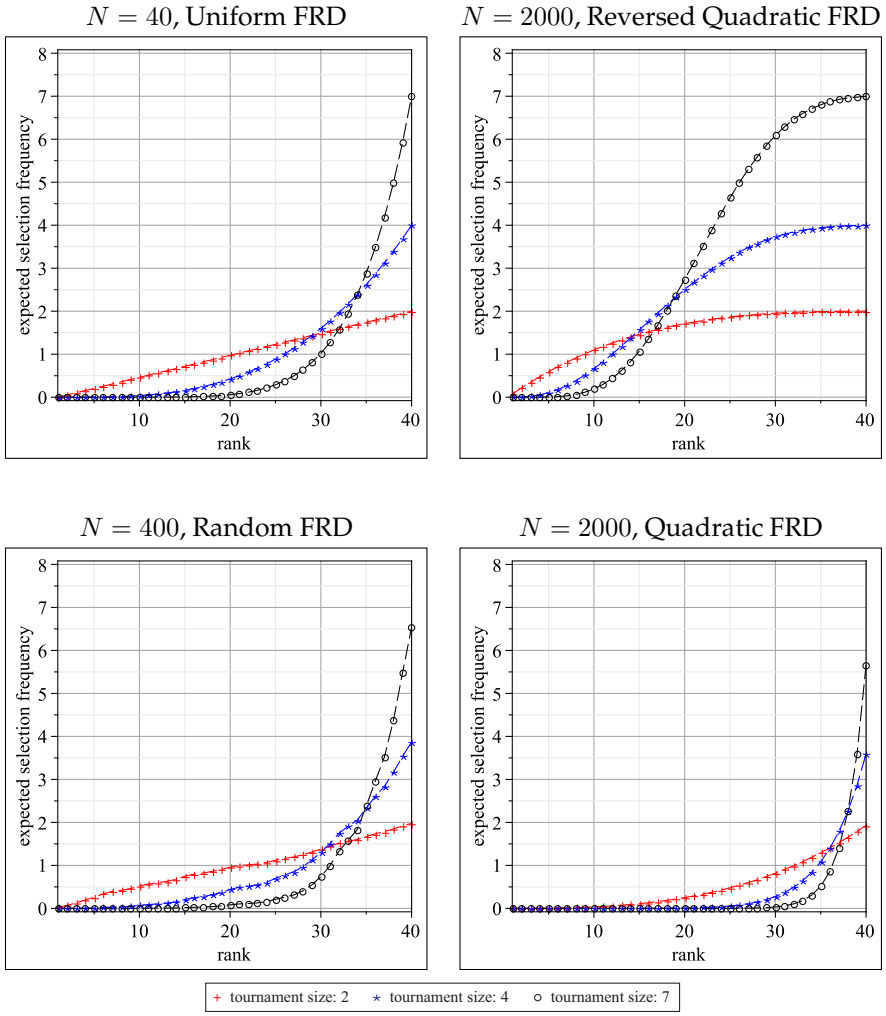


Figure 3.8: Selection frequency in the no-replacement tournament selection scheme on four populations with different FRDs.

With closer inspection of the total loss of program diversity measure, we observed that when larger tournament sizes are used, a slight difference occurs in the no-replacement tournament selection on the smaller-sized population (the top-left chart in Figures 3.4 and 3.7), whereas no noticeable difference exists on other-sized populations. This may be because in the no-replacement tournament selection, according to Equation 3.13, the probability that a program has never been sampled in $y = N$ tournaments is:

$$\left(\frac{N-k}{N}\right)^{N} = \left(\frac{\frac{N}{k}-1}{\frac{N}{k}}\right)^{\frac{N}{k}\,k} \approx e^{-k} \tag{3.19}$$

for large N/k. This equation is approximately the same as that in the standard tournament selection. However, for the smaller-sized population when larger

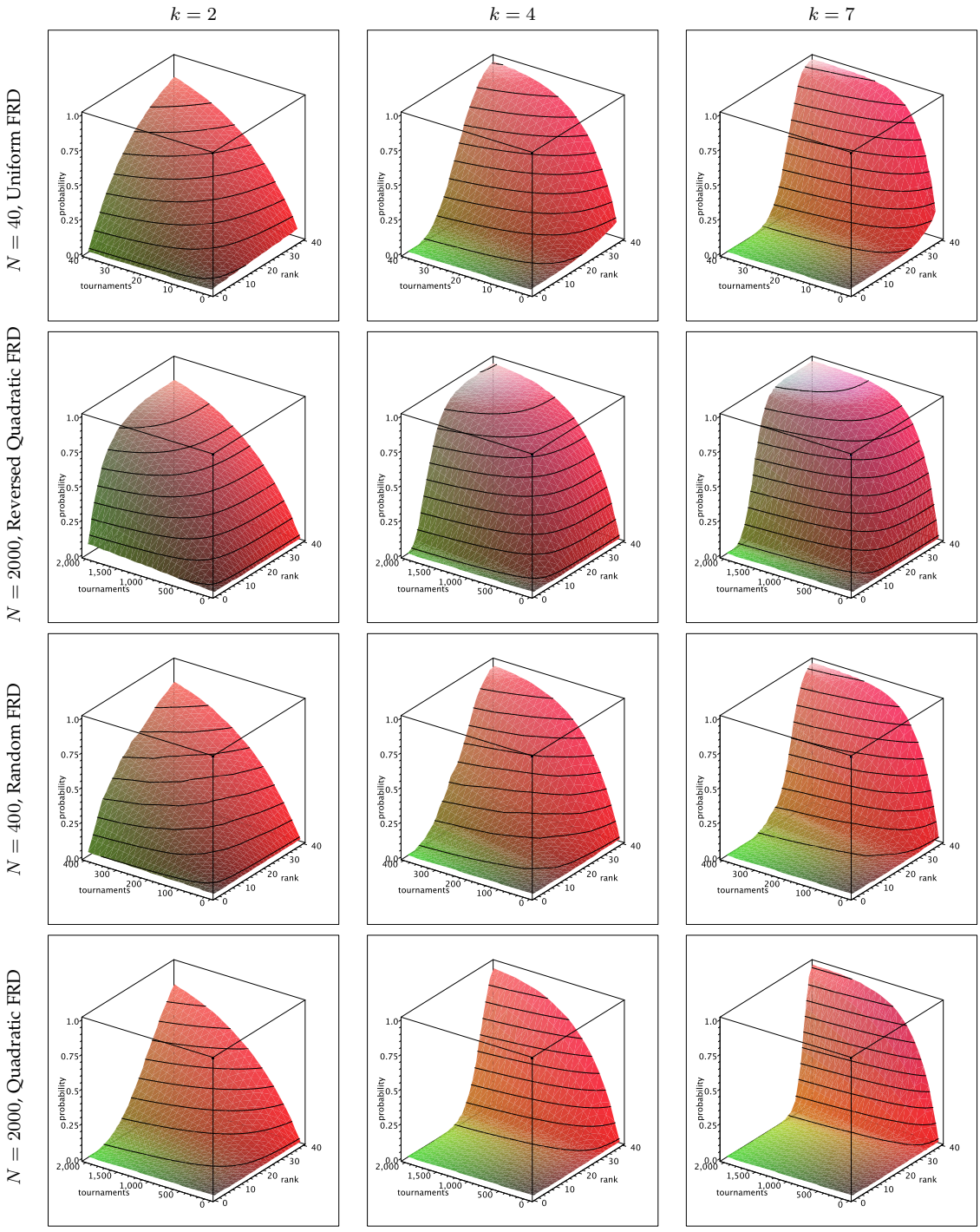


Figure 3.9: Selection probability distribution in the no-replacement tournament selection scheme with tournament size 2, 4 and 7 on four populations with different FRDs.

tournament sizes are used, this approximation is not valid. Therefore, the no-replacement tournament selection strategy does not help the loss of program diversity, especially when the size of a population is large.

Similar observations can be obtained by comparing the other two selection

pressure measures. The results show that if common tournament sizes and population sizes are used, no significant difference in selection behaviour has been observed between the two tournament selection schemes. Therefore, the next subsection examines the sampling behaviour to explore the underlying reasons.

3.5.4 Sampling behaviour analysis

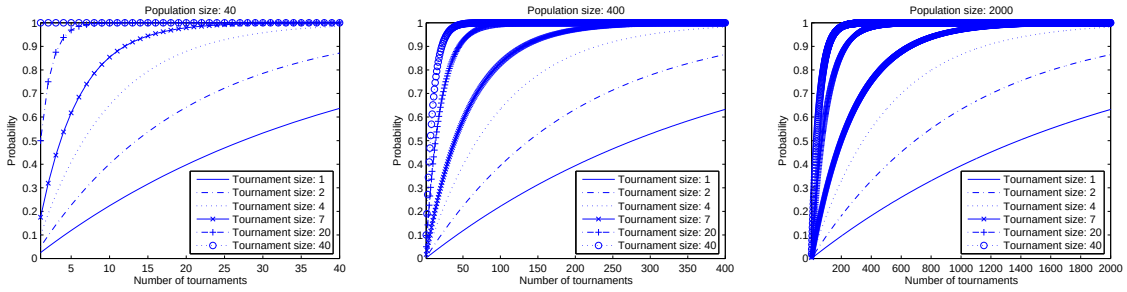


Figure 3.10: Trends of the probability that a program is sampled at least once in the no-replacement tournament selection in the selection phase. (Note that the scales on the x-axes differ.)

Figure 3.10 demonstrates the sampling behaviour in the no-replacement tournament selection via the probability trends of a program being sampled using six tournament sizes in three populations as the number of tournaments increases up to the corresponding population size. By comparing Figure 3.3 on page 52 and Figure 3.10, apart from the case of population size 40 and tournament size 40, which produces the 100% sampling probability in the no-replacement tournament selection, there are no noticeable differences between corresponding trends in the standard and no-replacement tournament selection schemes. The results are not surprising since both Equations (3.1) and (3.13) can be approximated by $1 - e^{-k\frac{y}{N}}$ for large N.

3.5.5 Significance in similarity or difference analysis

To further investigate the similarity or difference between the sampling behaviour in the two tournament selection schemes, we ask the following question: for a given population of size N, if we keep sampling individuals with replacement, then what is the largest number of sampling events at a certain level of confidence

that there will be no duplicates amongst the sampled individuals? Answering this question requires an analysis of the relationship between confidence level, population size and tournament size. Equation 3.20 models the relationship between the three factors, where N^k is the total number of different sampling results when sampling k individuals with replacement, $\frac{N!}{(N-k)!}$ is the number of sampling events such that no duplicate is in the k sampled individuals, and $(1-\alpha)$ is the confidence coefficient[1].

$$\frac{N!}{N^k\,(N-k)!} \geq 1-\alpha. \tag{3.20}$$

Figure 3.11 illustrates the relationship between population size N, tournament size k, and the confidence level. For instance, sampling 7 individuals with replacement will not sample duplicates with 99% confidence when the population size is about 2000, and 95% confidence when the population size is about 400, but only 90% confidence when the population size is about 200. We also calculated that when the population size is 40, the confidence level is only about 57% for $k = 7$. These results explained why we have observed only differences between the two tournament selection schemes on the smaller-sized population using larger tournament sizes.

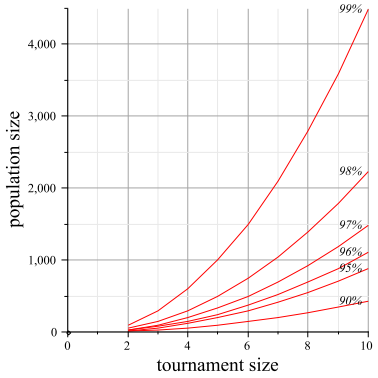


Figure 3.11: Confidence level, population size and tournament size. Note that tournament size is discrete but the plot shows curves to aid interpretation.

The results show that for common tournament sizes 4 or less, we would not expect to see any duplicates except for very small populations. Even for tournament size 7, we would expect only to see a small number of duplicates for

[1] α is *significance level* and $100(1-\alpha)\%$ is the confidence level.

populations less than 200 with 90% confidence. For most common settings of tournament sizes and population sizes, the multi-sampled issue *seldom* occurs in standard tournament selection. In addition, since duplicated individuals do not necessarily influence the result of a tournament when the duplicates have worse fitness values than other sampled individuals, the probability of significant difference between standard tournament selection and no-replacement tournament selection will be even smaller. Therefore eliminating the multi-sampled issue in standard tournament selection is very unlikely to significantly change the selection performance. As a result the multi-sampled issue generally is not crucial to the selection behaviour in standard tournament selection.

Given the difficulty of implementing sampling-without-replacement in a parallel architecture, most researchers have abandoned sampling-without-replacement, and used the simpler sampling-with-replacement scheme, hoping that the multi-sampled issue is not important. The results of our analysis justified this choice.

3.6 Analysis of the Not-Sampled Issue

The not-sampled issue aggravates the loss of program diversity. However, it is not clear how seriously it affects GP search. This section shows that the not-sampled issue is insignificant as well.

An obvious way to tackle the not-sampled issue is to increase the tournament size because larger tournament sizes provide a higher probability of an individual being sampled. However, increasing tournament size will increase the tournament competition level, and the loss of diversity contributed by not-selected individuals will increase, possibly resulting in even worse total loss of diversity.

The not-sampled issue will be completely solved only if every individual in a population is guaranteed to be sampled at least once during the selection phase. However, the sampling-*with*-replacement method in standard tournament selection cannot guarantee this no matter how other aspects of selection are changed. Therefore, a sampling-*without*-replacement strategy must be used for this purpose. One strategy is the no-replacement tournament selection method. Un-

fortunately, it still cannot solve the not-sampled issue unless we configure the tournament size to be the same as the population size. Obviously, applying the no-replacement tournament selection with such a configuration is not useful as it is effectively equivalent to always selecting the best of a population.

To investigate whether the not-sampled issue seriously affects the selection performance in the standard tournament selection, we will firstly develop an approach that satisfies the following requirements: (1) minimises the number of not-sampled individuals; (2) preserves the same tournament competition level as in the standard tournament selection; and (3) preserves selection pressure across the population at a level comparable to the standard tournament selection. We will then compare the approach with the standard tournament selection.

3.6.1 Different replacement strategies

A simple sampling-without-replacement strategy that solves the not-sampled issue is to return only the losers to the population at the end of each tournament. We termed this strategy *loser-replacement*. By using this strategy, the size of the population gradually decreases along the way to form the next generation. (At the end, the population will be smaller than the tournament size but these tournaments can be run at a reduced size.) The loser-replacement tournament selection will not have any selection pressure across the population. It will be very similar to a *random sequential selection* where every individual in the population can be randomly selected as a parent to mate but just once. The only difference between the outcomes of the loser-replacement tournament selection and the random sequential selection is the mating order. Although the loser-replacement strategy can ensure zero loss of diversity, it cannot preserve any selection pressure across population. Therefore, it is not very useful.

To satisfy all the essential requirements, we propose another sampling-without-replacement strategy. After choosing a winner, all sampled individuals are kept in a temporary pool instead of being immediately returned to the population. For this strategy, as long as the tournament size is greater than one, after a number of tournaments, the population will be empty and tournaments will stop. At that point, the population is refilled from the temporary pool to start a new *round* of

tournaments. More precisely, for a population S and tournaments of size k, the algorithm is:

1: Initialise an empty temporary pool T
2: **while** need to generate more offspring **do**
3: **if** $size(S) < k$ **then**
4: Refill: move all individuals from T to S
5: **end if**
6: Sample k individuals without replacement from the population S
7: Select the winner from the tournament
8: Move the k sampled individuals into T
9: **end while**

We term a tournament selection using this strategy as *round-replacement* tournament selection. The next subsections analyse this strategy to investigate the impact of the not-sampled issue.

3.6.2 Modelling round-replacement tournament selection

Assume N is a multiple of k, then after N/k tournaments, the population becomes empty. The round-replacement algorithm needs to refill the population to start another round of tournaments. There will be k rounds in total in order to form an entire next generation. It is obvious that any program will be sampled exactly k times during the selection phase thus there is no need to model the sampling probability. The selection probability is given in Lemma 3.

Lemma 3. *For a particular program $p \in S_j$, if W_j is the event that p wins or is selected in a tournament of size k, the probability of W_j is:*

$$P(W_j) = \frac{\sum_{n=1}^{k} \frac{1}{n} \begin{pmatrix} |S_j| - 1 \\ n - 1 \end{pmatrix} \begin{pmatrix} \sum_{i=1}^{j-1} |S_i| \\ k - n \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix}} \tag{3.21}$$

Proof. The characteristic of the round-replacement tournament selection is that it guarantees p will be sampled once in just one of the N/k tournaments in a round.

According to this, the effect of a full round of tournaments is to partition S into N/k disjoint subsets. The program p is a member of precisely one of these N/k subsets. Therefore the probability of it being *selected* in one tournament in a given round is exactly the same as in any other tournament in the same round. Furthermore, the probability of it being selected in one round is exactly the same as in any other round since all k rounds of tournaments are independent. Therefore we need only to model the selection probability of p in one tournament of one round. p could be selected if it is sampled in the tournament and no better-ranked programs are sampled in the same tournament; its selection probability will depend on the number of other programs having the same rank that are sampled in the same tournament.

Let E_j be the event that $p \in S_j$ is selected in a round of tournaments. The total number of ways of constructing a tournament containing the program p, $n-1$ other programs in the same S_j, and $k-n$ programs in $S_1, S_2, ..., S_{j-1}$ is[2]:

$$\sum_{n=1}^{k} \begin{pmatrix} |S_j|-1 \\ n-1 \end{pmatrix} \begin{pmatrix} \sum_{i=1}^{j-1}|S_i| \\ k-n \end{pmatrix} \tag{3.22}$$

As each of the n programs has an equal probability to be chosen as the winner, and there are $\begin{pmatrix} N-1 \\ k-1 \end{pmatrix}$ ways of constructing a tournament containing p, the probability of E_j is:

$$P(E_j) = \frac{\sum_{n=1}^{k} \frac{1}{n} \begin{pmatrix} |S_j|-1 \\ n-1 \end{pmatrix} \begin{pmatrix} \sum_{i=1}^{j-1}|S_i| \\ k-n \end{pmatrix}}{\begin{pmatrix} N-1 \\ k-1 \end{pmatrix}} \tag{3.23}$$

Since there are N/k tournaments in a round and the program p has an equal probability to be selected in any one of the N/k tournaments, the probability of W_j is:

$$P(W_j) = \frac{P(E_j)}{N/k} \tag{3.24}$$

thus we obtain Equation (3.21). □

[2]Assuming $\binom{a}{b} = 0$ if $b > a$.

Let $T_{j,c}$ be the event that p is selected at least once by the end of cth round. As the selection behaviour in any two rounds are independent and identical, the probability of $T_{j,c}$ is:

$$P(T_{j,c}) = 1 - (P(\overline{E_j}))^c \tag{3.25}$$

This equation together with Equation 3.21 will be used to calculate the selection probability distribution for the round-replacement tournament selection.

3.6.3 Selection behaviour analysis

The loss of program diversity, the selection frequency, and the selection probability distribution for the round-replacement tournament selection are illustrated in Figures 3.12, 3.13, and 3.14, respectively.

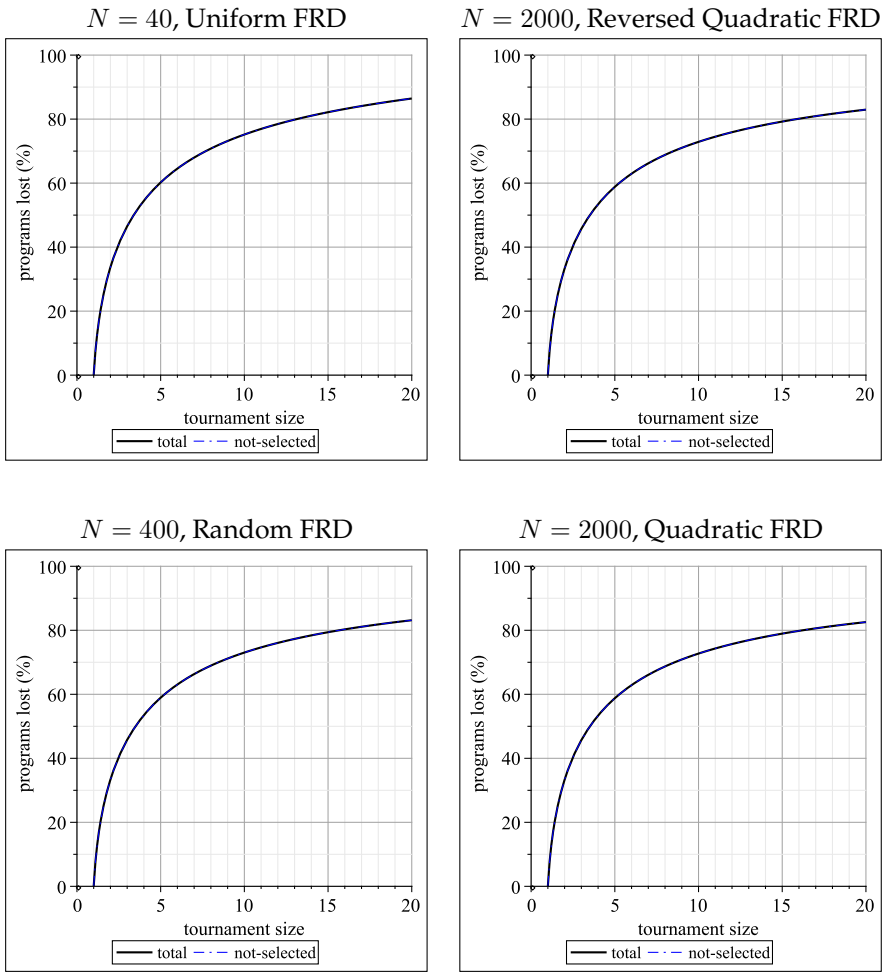


Figure 3.12: Loss of program diversity in the round-replacement tournament selection scheme on four populations with different FRDs. Note that tournament size is discrete but the plots show curves to aid interpretation.

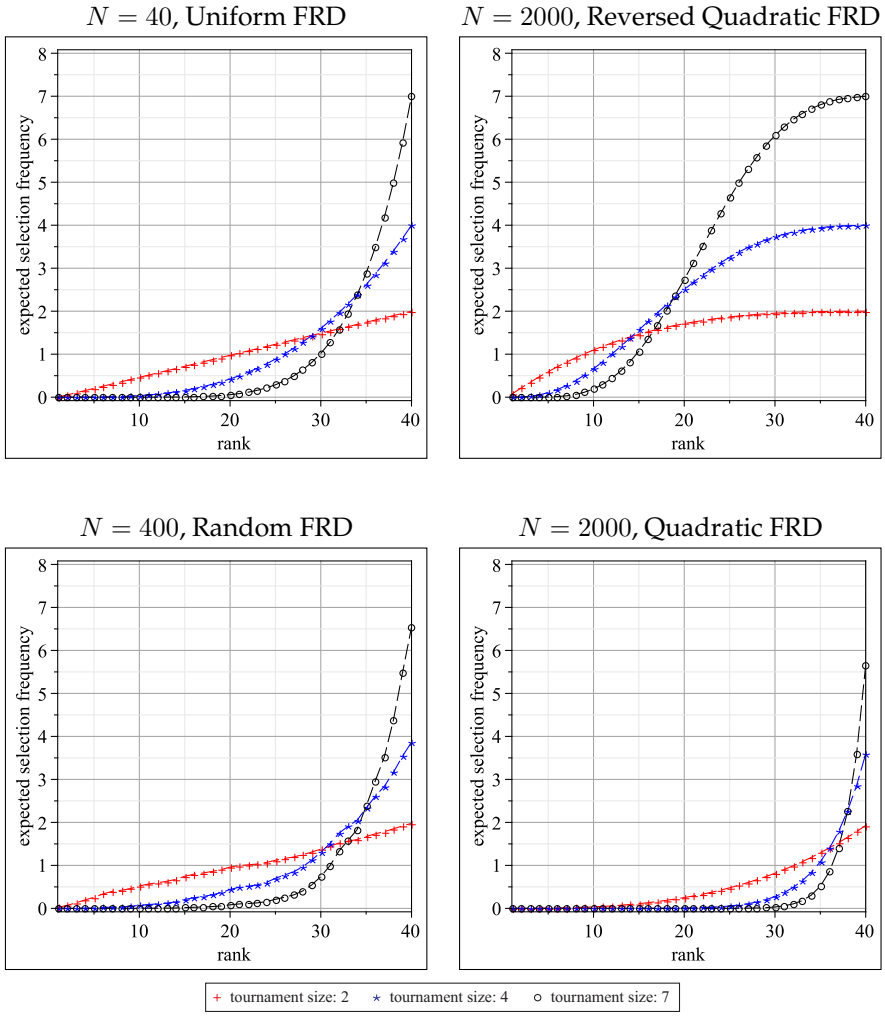


Figure 3.13: Selection frequency in the round-replacement tournament selection scheme on four populations with different FRDs.

In Figure 3.12, the trends of the total loss of diversity is identical to the contribution from the not-selected individuals because individuals are guaranteed to be sampled: precisely sampled once in a round and k times in total. Therefore, the round-replacement tournament selection minimises the loss of program diversity contributed by not-sampled individuals while maintaining the same tournament competition level as that in the standard tournament selection. Again there are no noticeable differences between the loss of program diversity measures on different-sized populations with different FRDs.

The loss of program diversity is significantly smaller with the round-replacement tournament selection than with the standard one for small tournament sizes ($k < 4$) in all population, but slightly larger for large tournament sizes in the smaller-sized population.

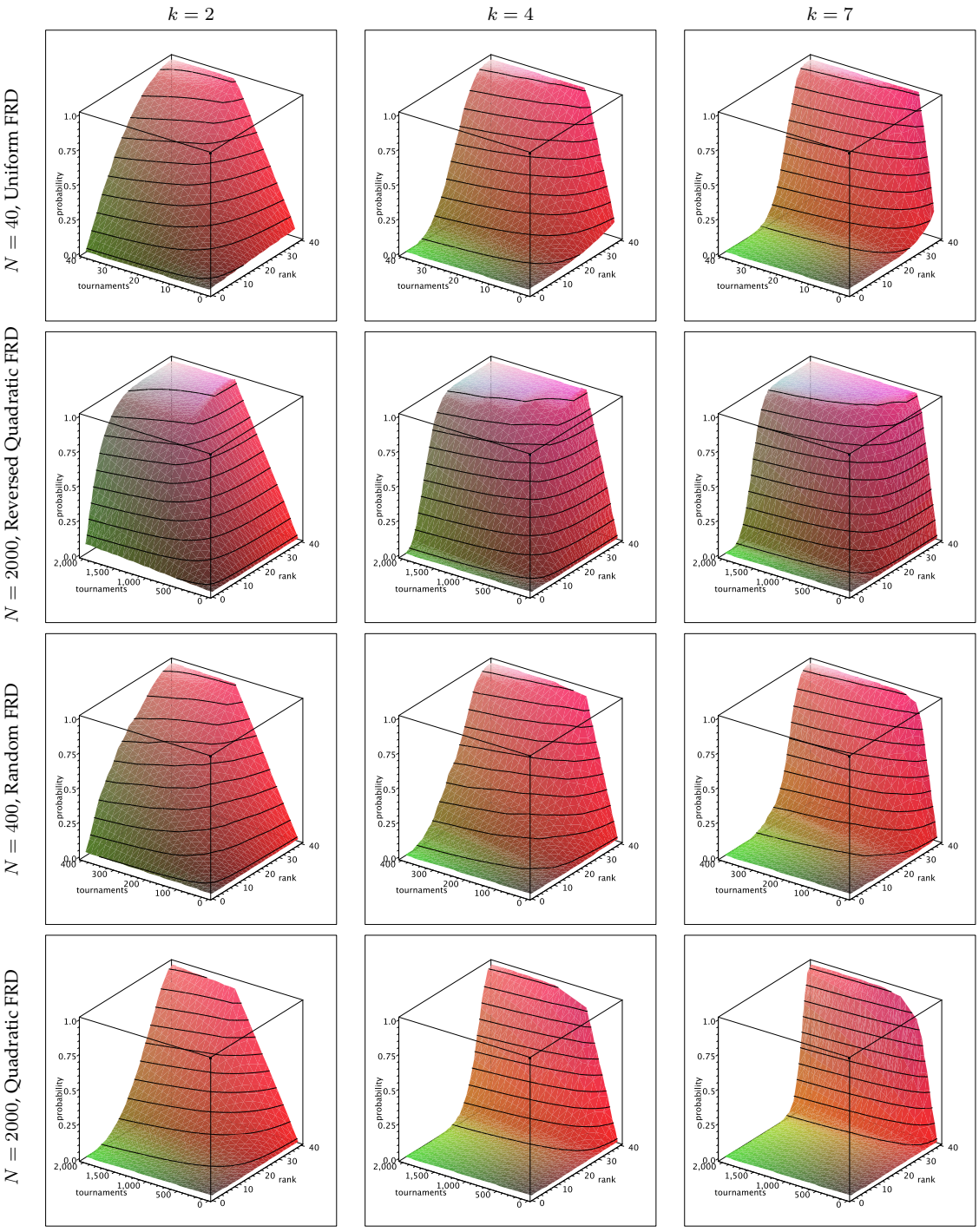


Figure 3.14: Selection probability distribution in the round-replacement tournament selection scheme with tournament size 2, 4 and 7 on four different FRDs.

From Figure 3.13, the trends of the selection frequency across each population are still very similar to the corresponding ones in the standard tournament selection. There is a slight difference in the smaller-sized population. Surprisingly, we find that Figure 3.13 seems to be identical to Figure 3.8 of the no-replacement tournament selection. In fact Equations 3.16 and 3.21 are mathematically equiva-

lent. The proof can be found in Appendix A.

While the selection frequency is the same in the no-replacement and round-replacement tournament selections, our selection probability distribution measure reveals the differences. Figure 3.14 shows that the round-replacement tournament selection has some different behaviour from the standard tournament selection and also from the no-replacement one, especially when the tournament size is 2. The differences are related to the top ranked individuals, whose selection probabilities reach 100% very quickly.

The fact that the selection frequency is identical in the no-replacement and the round-replacement tournament selections but the selection probability distribution is different shows that selection frequency sometimes is not adequate for distinguishing selection behaviour.

To further investigate whether the different selection behaviour in the round-replacement tournament selection can improve the GP search significantly, the next subsections present an experimental analysis of some common problems.

3.6.4 Experiment design

It is clear that inappropriate fitness functions will provide incorrect information to selection mechanisms and seriously affect their functionality. In order to reduce the side effect of inappropriate fitness functions, this thesis uses only problems from domains where fitness functions are well known.

3.6.4.1 data sets

The experiments involve three different problem domains: an Even-n-Parity problem (EvePar), a Symbolic Regression problem (SymReg), and a Binary Classification problem (BinCla) with increasing difficulties. We chose these three types of problems in particular because they have received considerable attention as examples in the literature of GP.

EvePar considers the case of $n = 6$. Therefore, there are 2^6 combinations of unique 6-bit length strings as fitness cases. SymReg is shown in Equation (3.26) and visualised in Figure 3.15. We generated 100 fitness cases by choosing 100

values for x from [-5,5] with equal steps. For EvePar and SymReg, all fitness cases are used for training, that is, the test data set is the same as the training data set.

$$f(x) = exp(1-x) \times sin(2\pi x) + 50sin(x) \tag{3.26}$$

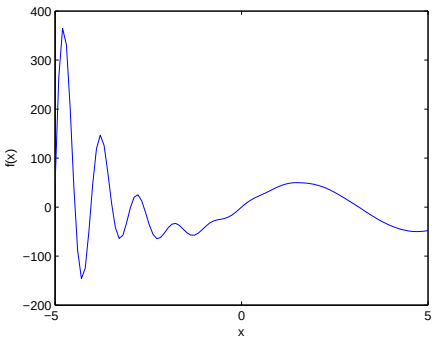


Figure 3.15: The symbolic regression problem.

BinCla involves determining whether examples represent a *malignant* or a *benign* breast cancer. The dataset is the Wisconsin Diagnostic Breast Cancer dataset chosen from the UCI Machine Learning repository [131]. BinCla consists of 569 data examples, where 357 are benign and 212 are malignant. It has 10 numeric measures (see Table 3.1) computed from a digitised image of a fine needle aspirate of a breast mass and are designed to describe characteristics of the cell nuclei present in the image. The mean, standard error, and "worst" of these measures are computed, resulting in 30 features [131]. The whole original data set is split randomly and equally into a training data set, a validation data set, and a test data set with class labellings being evenly distributed across the three data sets for each individual GP run.

Table 3.1: Ten features in the dataset of BinCla

a	radius	f	compactness
b	texture	g	concavity
c	perimeter	h	concave points
d	area	i	symmetry
e	smoothness	j	fractal dimension

3.6.4.2 function sets and terminal sets

The function set used for EvePar consists of the standard Boolean operators { *and*, *or*, *not* } and *if* function. The *if* function takes three arguments and returns its second argument if the first argument is *true*, and otherwise returns its third argument. In order to increase the problem difficulty, we do not include the *xor* function in the function set.

The function set used for SymReg includes the standard arithmetic binary operators { +, -, *, / } and unary operators { *abs*, *sin*, *exp* }. The / function returns zero if it is given invalid arguments.

The function set used for BinCla includes the standard arithmetic binary operators { +, -, *, / }. We hypothesised that convergence might be quicker if using only the four arithmetic operators, and more functions might lead to better results. Therefore, the function set also includes unary operators { *abs*, *sqrt*, *sin* } and *if* function. The *sqrt* function automatically converts a negative argument to a positive one before operating on it. The *if* function takes three arguments and returns its second argument if the first argument is positive, and returns its third argument otherwise. The *if* function allows a program to contain a different expression in different regions of the feature space, and allows discontinuous programs, rather than insisting on smooth functions.

The terminal set for EvePar consists of n boolean variables. The terminal set for SymReg and BinCla includes a single variable x and 30 terminals, respectively. Real valued constants in the range [-5.0, 5.0] are also included in the terminal sets for SymReg and BinCla. The probability mass assigned to the whole range of constants when constructing programs is set to 5%.

3.6.4.3 fitness function

For even-n-parity problems, the standard fitness function counts the number of wrong outputs (misses) for the 2^n combinations of n-bit strings and treats zero misses as the best raw fitness [93]. There is an issue with this fitness function: the worst program according to this fitness function is the one that has 2^n misses. However, this program actually captures most of the structure of the problem

and can be easily converted to a program of zero misses by adding a *not* function node to the root of the program. Therefore, programs with a very large number of misses are, in a sense, just as good as programs with very few misses.

In this thesis, we used a new fitness function for EvePar:

$$fitness = \begin{cases} m & , \quad if\ m < 2^{n-1} \\ 2^n - m & , \quad otherwise \end{cases} \tag{3.27}$$

where m is the number of misses.

The fitness function in SymReg is the root-mean-square (RMS) error of the outputs of a program relative to the expected outputs. Because neither class is weighted over the other, the fitness function for BinCla is the classification error rate on the training data set (the fraction of fitness cases that are incorrectly classified by a program as a proportion of the total number of fitness cases in the training data set). A program classifies the fitness case as *benign* if the output of the program is positive, and *malignant* otherwise. Note that class imbalance design in fitness function for BinCla is beyond the scope of this thesis. All three problems have an ideal fitness of zero.

3.6.4.4 genetic parameters and configuration

The genetic parameters are the same for all three problems. The ramped half-and-half method is used to create new programs and the maximum depth of creation is four (counted from zero). To prevent code bloat, the maximum size of a program is set to 50 nodes during evolution based on some initial experimental results. The crossover rate, the mutation rate, and the copy rate are 85%, 10% and 5% respectively. The best individual in the current generation is explicitly copied into the next generation, ensuring that the population does not lose its previous best solution[3]. A run is terminated when the number of generations reaches the pre-defined maximum of 101 (including the initial generation), or the problem has been solved (there is a program with a fitness of zero on the training data set), or the error rate on the validation set starts increasing (for BinCla). Three tournament sizes 2, 4, and 7 are used. Consequently, the population size is set to

[3]This is referred as elitism [149].

504 in order to have zero remainder at the end of a round of tournaments in the round-replacement tournament selection.

We ran experiments comparing the two GP systems using the standard and the round-replacement tournament selections respectively for each of the three problems. In each experiment, we repeated the whole evolutionary process 500 times independently. In each pair of the 500 runs, an initial population is generated randomly and is provided to both GP systems in order to reduce the performance variance caused by different initial populations.

3.6.5 Experimental results and analysis

Table 3.2 compares the performances of GP systems using the standard and the round-replacement tournament selection schemes. The measure for EvePar is the failure rate, measuring the fraction of runs that were not able to return the ideal solution. The best value is zero percent, meaning every run is successful. The measures for SymReg and BinCla are the averages of the RMS error and the classification error rate on test data over 500 runs respectively, thus the smaller the value, the better the performance. Note that the standard deviation is shown after the $\pm$ sign.

Table 3.2: Performance comparison between the round-replacement and the standard tournament selection schemes.

Tournament Selection		EvePar	SymReg	BinCla
Scheme	Size	Failure (%)	RMS Error	Test Error Rate (%)
round-replacement	2	99.6	47.4 ± 5.3	8.4 ± 2.7
	4	79.4	38.3 ± 8.0	8.6 ± 2.6
	7	77.6	40.6 ± 11.4	8.8 ± 2.7
standard	2	100	48.2 ± 5.2	9.2 ± 2.9
	4	80.6	37.6 ± 8.3	8.7 ± 2.7
	7	82.4	40.9 ± 11.3	8.7 ± 2.7

The results demonstrate that the round-replacement tournament selection has some advantages. In order to provide statistically sound comparison results, we calculated the confidence intervals at 95% and 99% levels (two-sided) for the differences in failure rates, in RMS errors, and in error rates for EvePar, SymReg and

BinCla respectively. For EvePar, we used the formula

$$\hat{P}_1 - \hat{P}_2 \pm Z\sqrt{\hat{P}_1(1-\hat{P}_1)/500 + \hat{P}_2(1-\hat{P}_2)/500} \tag{3.28}$$

where $\hat{P}_1$ is the failure rate using the round-replacement tournament selection, $\hat{P}_2$ is the failure rate using the standard tournament selection, and Z is 1.96 for 95% confidence and 2.58 for 99% confidence [20]. For SymReg and BinCla, we firstly calculated the difference of the measures between a pair of runs using the same initial population for each of the 500 pairs of runs, then used the formula

$$\bar{x} \pm Z\frac{s}{\sqrt{500}} \tag{3.29}$$

to calculate the confidence interval, where $\bar{x}$ is the average difference over 500 values and s is the standard deviation [20]. If zero is not included in the confidence interval, then the difference is statistically significant [20].

Table 3.3 shows the confidence intervals only at the 95% level, since the statistical analysis results from the two levels are consistent. Significant differences (either better or worse) are shown in bold. According to the performance measures, the round-replacement tournament selection is better than the standard one when the confident interval is less than zero.

Table 3.3: Confidence intervals for differences in performance at 95% level.

Tournament size	EvePar	SymReg	BinCla
2	(-0.95, 0.15)	**(-1.48, -0.24)**	**(-1.05, -0.43)**
4	(-6.16, 3.76)	(-0.22, 1.57)	(-0.32, 0.24)
7	(-9.75, 0.15)	(-1.47, 0.85)	(-0.25, 0.32)

From the table, for tournament size 2 and for SymReg and BinCla problems, the improvement of the round-replacement tournament selection is statistically significant. This observation is similar to the conclusions of Sokolov and Whitley [173]. However, practically the differences are small.

For tournament sizes 4 and 7, there are no statistically significant differences between the round-replacement and standard tournament selections. This is because only 1.8% and 0.09% of the population are not-sampled respectively in the

standard tournament selection (from Equation 3.1). There is little impact on the overall performance from the slight differences on the selection probability of the top-ranked programs.

We also compared the best performance of the round-replacement tournament selection with the best performance of the standard one for SymReg and BinCla; the differences were not statistically significant. A possible explanation is that although every program can be sampled in the round-replacement tournament selection, not all of these extra sampled programs can win tournaments. In addition, the number of extra programs which won the tournaments do not necessarily contribute to evolution. Therefore, the overall contribution to the search performance from these extra sampled programs would be limited.

Sokolov and Whitley's findings [173] suggested that performance could be improved by addressing the not-sampled issue in GA. Our experiments confirmed this in GP for some data sets and showed that the improvement was statistically significant, though not large. However, Sokolov and Whitley considered only tournament size 2. Our experiments included larger tournament sizes and showed that there was no statistically significant improvement for the larger tournament sizes. Furthermore, the performance of larger tournament sizes with the standard tournament selection was as good as or better than the performance of tournament size 2 with the round-replacement tournament selection. Therefore, there is no advantage in explicitly addressing the not-sampled issue.

The analysis results show that although the not-sampled issue can be solved, overall the different selection behaviour provided by the round-replacement tournament selection alone appears to be unable to significantly improve a GP system for the given tasks. The not-sampled issue does not *seriously* affect the selection performance in the standard tournament selection.

3.7 Analysis of the High Between-Group Selection Pressure Issue

Different sampling-without-replacement strategies appear to have little influence on the selection behaviour in standard tournament selection. They also have no

effect on the high between-group selection pressure issue, which causes the parent selection process to be dominated by a group of very similar programs and genetic diversity to be reduced prematurely. None of the standard, no-replacement, or round-replacement tournament selection methods can adjust selection bias in response to the FRD of a population. Since in general they all use a fixed tournament size, a skewed FRD actually aggravates the selection bias (as discussed on page 58). What is required in these circumstances is a reduction in selection pressure to allow low ranked programs to be selected to maintain the genetic diversity.

This is just part of a more general issue: the evolutionary learning process itself is very dynamic. At some stages, it requires a fast convergence rate (i.e., high parent selection pressure) to find a solution quickly; at other stages, it requires a slow convergence rate (i.e., low parent selection pressure) to avoid being confined to a local maximum. These requirements could be achieved by changing tournament size dynamically in standard tournament selection. However, standard tournament selection is not aware of the dynamic requests. In order to pick the correct tournament size, it should collaborate with an extra component that can reveal the underlying dynamics and determine the requests. However, such a component has not been seen so far.

To address these issues, we need to modify the standard tournament selection to become aware of the dynamics along evolution and to be able to adjust parent selection pressure accordingly. Since tournament selection uses fitness rankings to select parents, and population FRDs reflects evolutionary dynamics, we propose an automatic selection pressure control approach using the knowledge of the population FRD.

3.7.1 Clustering tournament selection

The proposed approach is called *clustering tournament selection*. Figure 3.16 gives an overview of the approach and shows the relationships between the major components: population clustering and clustering tournament selection. Other standard components of GP are not detailed in the figure.

In the approach, the first component is population clustering. Populations are

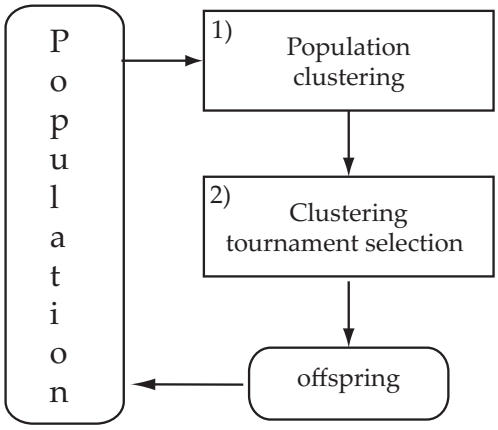


Figure 3.16: Overview and relationship between the major components.

clustered according to fitness values, and each cluster is then assigned a fitness value. The second component is a new tournament selection method called *clustering tournament selection*. Instead of sampling individuals as tournament candidates, the clusters are treated as the tournament candidates in the clustering tournament selection method: the best fitness cluster wins the tournament, and a program in the cluster is randomly selected to participate in the recombination process.

For a population S, which has been clustered into a set of $|S|$ clusters according to fitness values, the clustering tournament selection algorithm is as follows:

1: **for** $y = 1$ to N **do**
2: Sample k clusters from the $|S|$ clusters with replacement
3: Select the winning cluster from the tournament based on the fitness values
4: Return an individual program randomly chosen from the winning cluster
5: **end for**

Because the number of clusters in each generation reflects the dynamic evolutionary process, especially the degree of convergence of the population, we expect the selection pressure can be automatically adjusted along evolution accordingly.

The next subsections will model and analyse selection behaviour of clustering tournament selection, followed by experiments on EvePar, SymReg and BinCla to demonstrate the effectiveness of clustering tournament selection.

3.7.2 Modelling clustering tournament selection

Lemma 4. *Let S_j be the cluster of individuals of rank j in the population. The probability of the event D that a program $p \in S_j$ is sampled at least once in a tournament of size k is*

$$P(D) = 1 - (1 - \frac{1}{|S||S_j|})^k \tag{3.30}$$

Proof. In contrast to the standard tournament selection schemes, the sampling behaviour in clustering tournament selection is influenced by the size of each cluster. It is clear that each cluster has the same probability $1/|S|$ to be sampled. Individuals in a cluster have equal probability of being sampled, $1/|S_j|$. Therefore, the probability that p is sampled is $\frac{1}{|S||S_j|}$. The probability that p is never sampled into a tournament of size k is $(1 - \frac{1}{|S||S_j|})^k$. Thus, we obtain Equation 3.30. □

Lemma 5. *Let S_j be the cluster of individuals of rank j in the population, the probability of the event E_j that a program $p \in S_j$ is selected in a single tournament is*

$$P(E_j) = \frac{(j)^k - (j-1)^k}{|S|^k \times |S_j|} \tag{3.31}$$

Proof. According to the algorithm, the number of tournament candidates is effectively reduced from the whole population size N to the number of clusters $|S|$. The probability that a cluster ranked j wins a tournament is simply:

$$\frac{(j)^k - (j-1)^k}{|S|^k} \tag{3.32}$$

Since all individuals in the winning cluster have the same probability to be chosen as a parent, we divide Equation (3.32) by the size of the jth cluster $|S_j|$ and obtain Equation (3.31). □

3.7.3 The loss of program diversity analysis

Figure 3.17 illustrates the loss of program diversity of the clustering tournament selection on four populations with different FRDs.

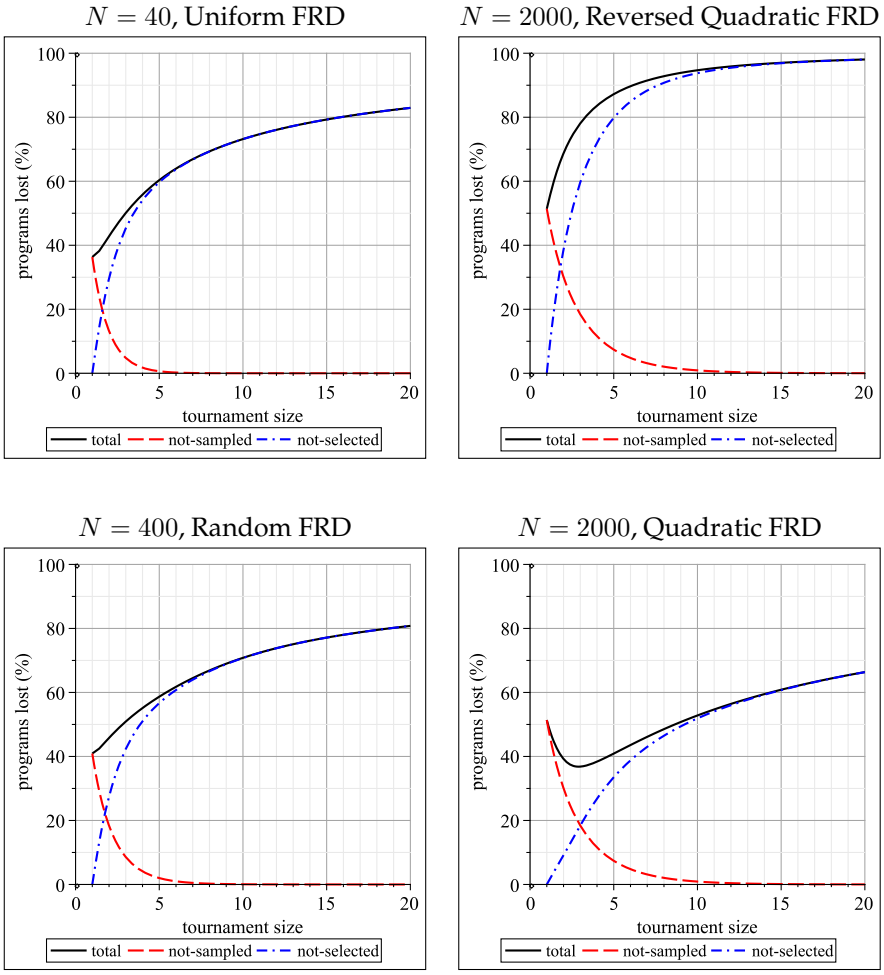


Figure 3.17: Loss of program diversity in the clustering tournament selection scheme on four different FRDs. Note that tournament size is discrete but the plots show curves to aid interpretation.

In the clustering tournament selection, for the uniform FRD, the three loss of program diversity measures are identical to those of the standard tournament selection (see Figure 3.4). This is because each cluster contains the same number of individuals (in this case the number is one) so that the clustering tournament selection is effectively acting the same as the standard tournament selection.

For the reversed quadratic FRD, the total loss of program diversity is considerably higher compared with that of the standard tournament selection and compared with those for other FRDs. We expect that the lost programs are mainly the worse-ranked individuals. By ignoring most of the worse-ranked individuals at this stage, the GP search will be able to concentrate on the promising region so that the evolution will speed up to save unnecessary cost. The next subsection will verify the expectation when analysing the selection frequency.

For the random FRD, there are only slight differences when comparing with that in the standard tournament selection.

For the quadratic FRD, the total loss of program diversity in the clustering tournament selection is greater than that in the standard one when the tournament size is one, but is considerably lower for other tournament sizes. The reduction quickly reaches by about 20% ($60\% - 40\% = 20\%$) when the tournament size increases to five. Also we observe that when the tournament size is 3, the total loss of program diversity becomes the lowest. The figure indicates that the program diversity is maintained in a better manner than that in the standard tournament selection. It is also what we expected for this type of FRD, as it may slow down the population convergence to avoid the confinement to local optima.

3.7.4 The selection frequency and the selection probability distribution analyses

Figures 3.18 and 3.19 illustrate the selection frequency and the selection probability distribution of clustering tournament selection on the four populations with different FRDs. Three tournament sizes are used to demonstrate the influences from different tournament sizes on the impacts of the clustering tournament selection. These figures show that the two measures provide consistent results. Therefore, this subsection discusses in detail only the results of the selection frequency as it is easier to understand results presented in 2D than in 3D.

Recall that the tournament size 3 provides the lowest total loss of program diversity for the quadratic FRD, therefore in addition to the usually used three tournament sizes, the tournament size 3 is added in this analysis and its impact is presented in a dash line in Figure 3.18.

The selection frequency trends on the uniform FRD in the clustering tournament selection are identical to those of the standard tournament selection for the reason given in Section 3.7.3.

The other three FRDs reveal significant differences when compared with the standard tournament selection (see Figure 3.5 on page 57).

For the reversed quadratic FRD (representing the early stage of evolution),

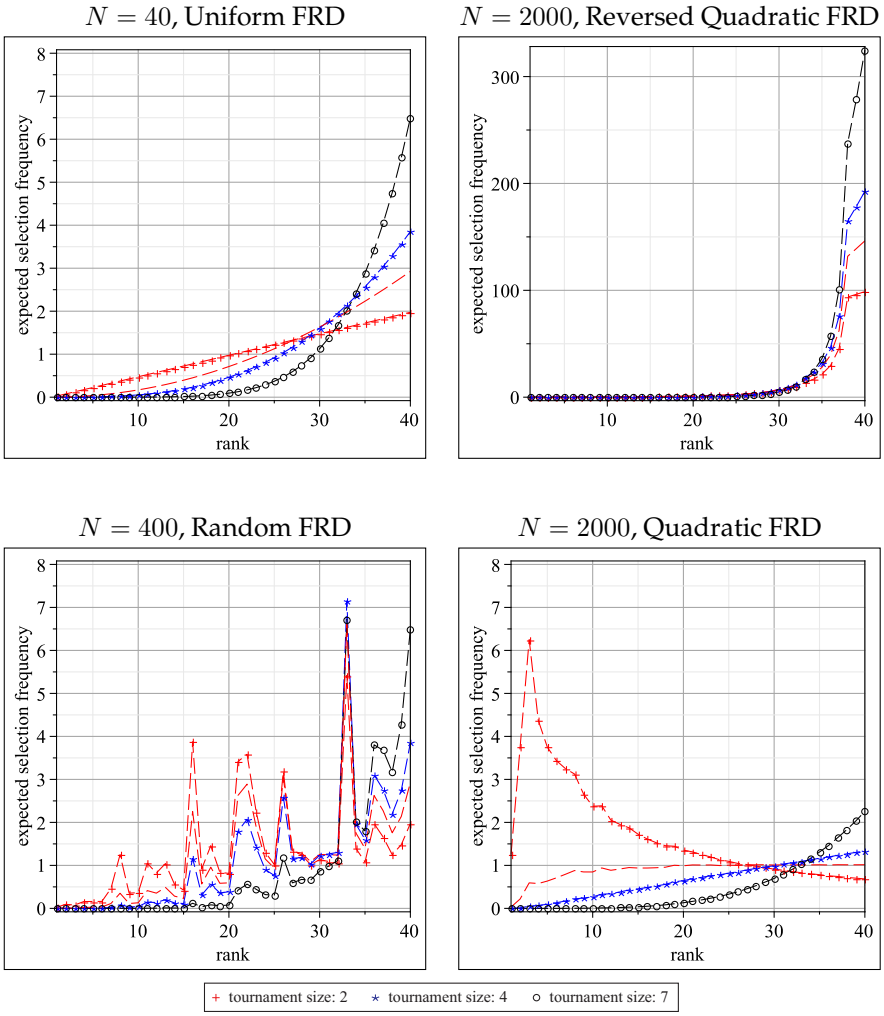


Figure 3.18: Selection frequency of the clustering tournament selection scheme on four populations with different FRDs. Note that the extra dash line represents tournament size 3.

most of the low fitness ranks have very low selection frequencies so that they are effectively discarded. This observation supports our expectation in the analysis of the loss of program diversity in the previous section and meets the desiderata of a good selection scheme.

For the random FRD (representing the middle stage of evolution), the selection frequency trends are very ragged instead of the smooth trends we usually saw in the standard, the no-replacement, and the round-replacement tournament selections. There is some interesting selection behaviour here. For instance, for the tournament size 4, the expected selection frequency for an individual program of rank 33 is above 7, while the expected selection frequencies for individuals of better ranks are much lower; even one of the best-ranked individuals in

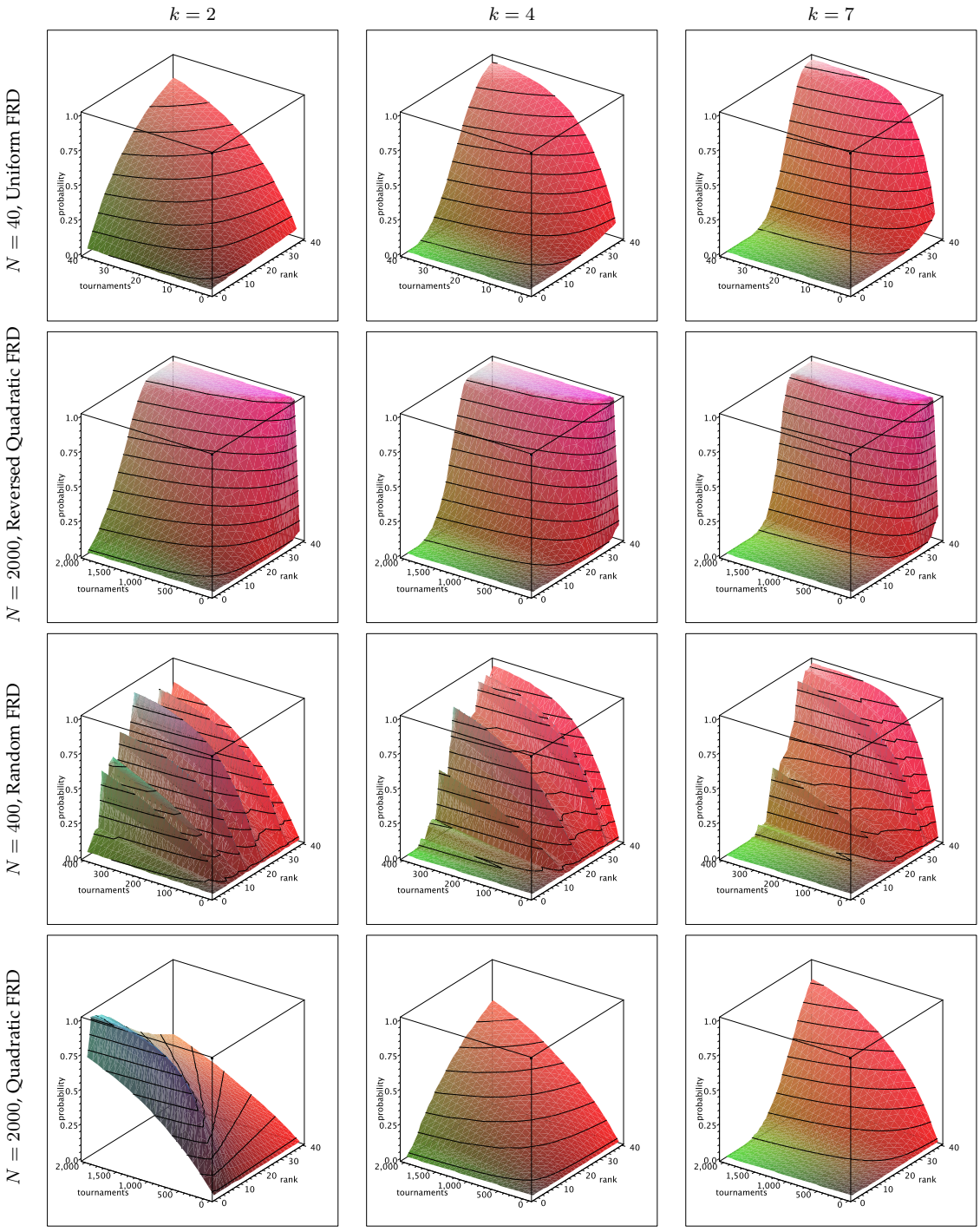


Figure 3.19: Selection probability distributions of the clustering tournament selection scheme with tournament size 2, 4 and 7 on four different FRDs.

the population is below 4. From Figure 3.2 on page 50, we can see that $|S_{33}|$ is only 3 while $|S_j|_{j>33}$ are much higher. The results show that apart from being governed by the tournament size, the clustering tournament selection is aware of the random changes in the FRD and can adjust the selection pressure automatically. It gives a relatively high selection preference to an individual in a fitness

bag with a smaller size to increase the chance of propagating its genetic material. It then gives relatively low selection preferences to other better individuals in fitness bags with larger sizes to restrict their propagation. This kind of selection behaviour is unique to the clustering tournament selection and appears to again meet the desiderata expectation of a good selection scheme (see the discussion on page 59).

For the quadratic FRD (representing a converged stage of evolution), the clustering tournament selection significantly reduces the selection frequency of better-ranked individuals, while increasing the frequency of middle-ranked individuals. Therefore, the clustering tournament selection can reduce the chance that groups of better-ranked individuals dominate the next generation and it is better able to maintain the population diversity than the standard one.

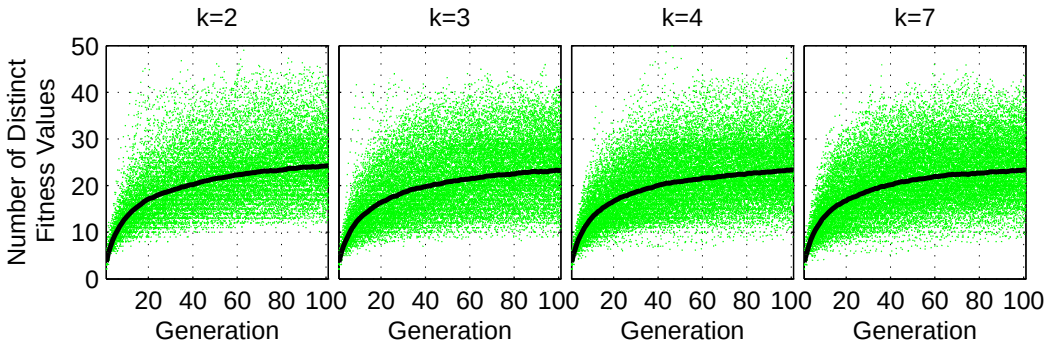
Note that for the quadratic FRD, tournament size 2 resulted in a strong bias to worse-ranked individuals, especially the third-ranked ones[4]; this may be undesirable. On the other hand, tournament size 3 provided almost even selection frequencies on all fitness ranks. This observation may explain why tournament size 3 provided the lowest total loss of program diversity.

In summary, the analysis results showed that in addition to the usual selection preference for better individuals governed by tournament size, the clustering tournament selection tends to give additional selection preference to individuals in small sized clusters. Furthermore, when most of the population are of worse fitness ranks and evolution encounters a danger of missing good individuals, it tends to increase selection bias to better individuals, hoping to quickly drive the population to promising regions. When the population tends to converge to local optima and evolution encounters a danger of losing genetic material, it tends to decrease selection bias to better individuals, hoping to keep the population diverse. Therefore, the clustering tournament selection is an automatically biased parent selection scheme that is needed by the dynamic evolutionary process.

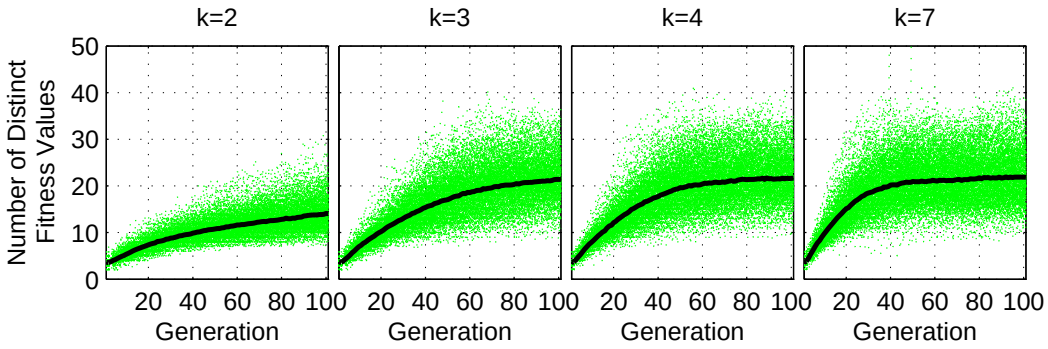
[4]This is because the ranks 1 to 3 have the same smallest number of individuals (Figure 3.2).

3.7.5 Impact on population diversity analysis

The simulations above suggest that the clustering tournament selection can be aware of the dynamics in evolution, and adjust the parent selection pressure accordingly. However, parent selection pressure is only one of the many factors influencing GP search, so the impact of the dynamic parent selection pressure adjustment needs to be experimentally tested. Therefore, we conducted sets of experiments based on the same set of problems and the same sets of configurations used in analysing the round-replacement tournament selection, but with an extra set of experiments using tournament size 3. This subsection analyses the impact of the clustering tournament selection on the population diversity in terms of the number of distinct fitness values; the next subsection analyses the impact on the overall GP search performance.



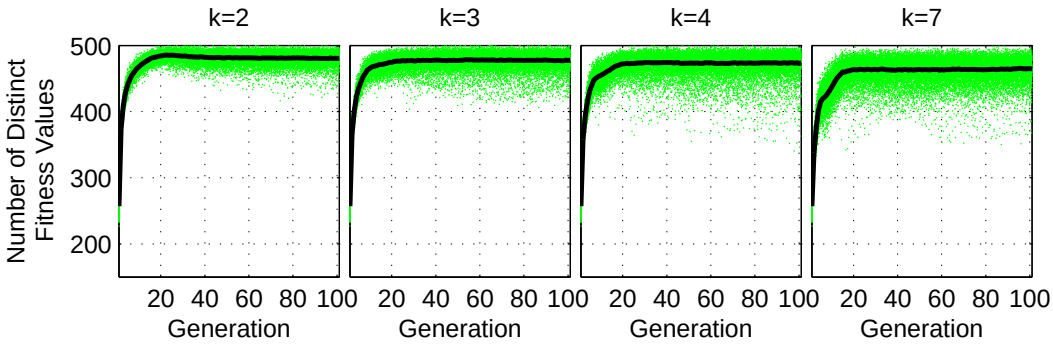
(a) clustering tournament selection for EvePar



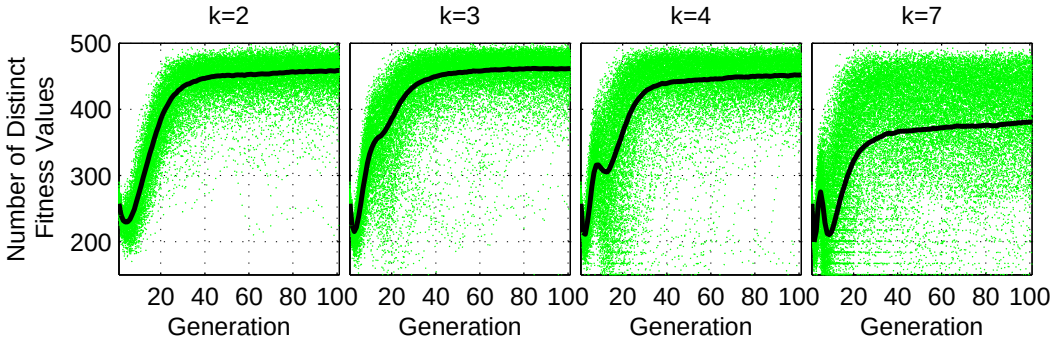
(b) standard tournament selection for EvePar

Figure 3.20: Comparison of population diversity maintenance between the clustering tournament selection and the standard tournament selection for EvePar for four tournament sizes.

Figures 3.20, 3.21 and 3.22 compare the clustering tournament selection and the standard tournament selection in terms of population diversity measured by



(a) clustering tournament selection for SymReg



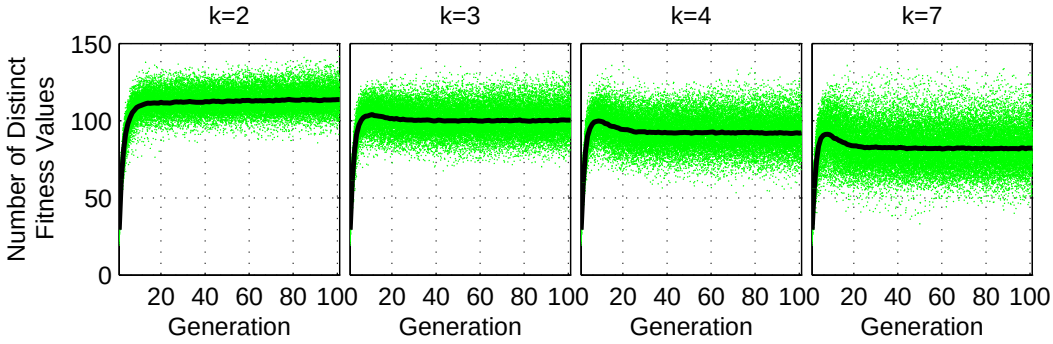
(b) standard tournament selection for SymReg

Figure 3.21: Comparison of population diversity maintenance between the clustering tournament selection and the standard tournament selection for SymReg for four tournament sizes.

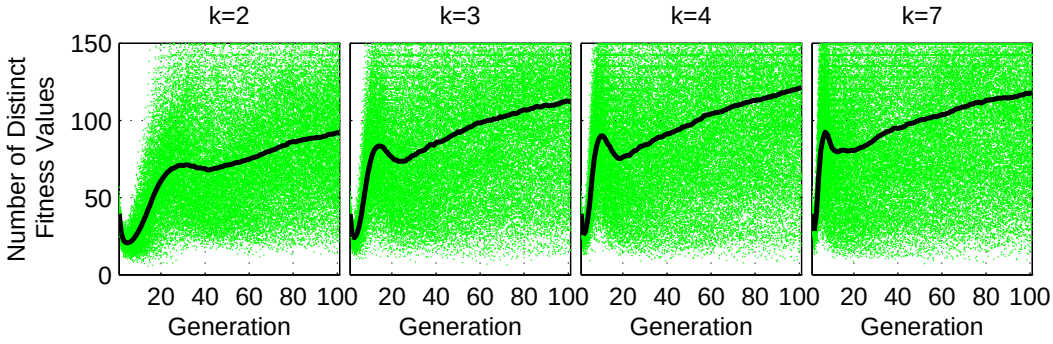
the number of distinct fitness values generation by generation using each of the four tournament sizes for EvePar, SymReg, and BinCla, respectively. The dark line in each chart represents the mean value over the 500 runs.

It is clear that the clustering tournament selection can quickly increase the population diversity to a certain level and maintain it stably. The four different tournament sizes have only small impact on the population diversity: for EvePar the four trends of the average numbers of distinct fitness values are almost identical, and for SymReg and BinCla there are only slight drops when the tournament size increases.

In contrast, the standard tournament selection performs differently, especially for SymReg and BinCla (chart (b) in Figures 3.21 and 3.22). The population diversity fluctuates along evolution and has larger variation in the 500 runs. It is also sensitive to tournament size. This comparison demonstrates the advantage of the clustering tournament selection in maintaining population diversity in terms of the number of distinct fitness values.



(a) clustering tournament selection for BinCla



(b) standard tournament selection for BinCla

Figure 3.22: Comparison of population diversity maintenance between the clustering tournament selection and the standard tournament selection for BinCla for four tournament sizes.

3.7.6 Overall GP search performance analysis

Table 3.4 compares the performances of GP systems using the standard and the clustering tournament selection schemes. Table 3.5 only shows the confidence intervals of the differences between the performances at 99% level since the statistical analysis at the 95% and 99% levels gives a similar pattern.

Table 3.4: Performance comparison between the clustering and the standard tournament selection schemes. (Some results for the standard tournament selection are repeated from Table 3.2 on page 80.)

Tournament Selection		EvePar	SymReg	BinCla
Scheme	Size	Failure (%)	RMS Error	Test Error Rate (%)
clustering	2	91.4	47.6 ± 5.9	7.4 ± 2.3
	3	87.2	39.7 ± 7.6	7.5 ± 2.3
	4	88.0	36.8 ± 7.9	7.7 ± 2.5
	7	88.8	33.5 ± 8.3	7.9 ± 2.5
standard	2	100	48.2 ± 5.2	9.2 ± 2.9
	3	87.0	39.9 ± 6.6	8.7 ± 2.7
	4	80.6	37.6 ± 8.3	8.7 ± 2.7
	7	82.4	40.9 ± 11.3	8.7 ± 2.7

Table 3.5: Confidence intervals at 99% level for the differences between the clustering and the standard tournament selection schemes.

Tournament size	EvePar	SymReg	BinCla
2	**(-11.83,-5.37)**	(-1.54, 0.25)	**(-2.11, -1.30)**
3	(-6.54, 4.14)	(-1.31, 0.91)	**(-1.57, -0.82)**
4	**(1.49,13.31)**	(-2.00, 0.47)	**(-1.40, -0.65)**
7	**(0.69,12.11)**	**(-8.87, -5.88)**	**(-1.25, -0.48)**

For BinCla (the hardest problem), the clustering tournament selection is consistently significantly better than the standard one for all four tournament sizes.

For SymReg, the clustering tournament selection is slightly better than the standard one using tournament sizes 2, 3, and 4, but significantly better only for tournament size 7. A large tournament size represents a strong selection bias towards better individuals and therefore there is a great potential for losing diversity. The clustering tournament selection appears to be able to counteract this potential effectively.

For EvePar (the simplest problem), when the tournament size is 2, the clustering tournament selection is significantly better than the standard tournament selection. However, when the tournament size is 4 or 7, it is significantly worse than the standard tournament selection.

The performance reported here shows that when the parent selection pressure is adjusted according to the dynamics in evolution and the population diversity is well maintained by the clustering tournament selection, the overall GP search performance is improved in most problems, but not every case. Possible explanations for the exceptions include:

- Easy problems can be solved easily using high selection pressure so that it is not necessary to adjust the parent selection pressure.
- Although good parents may be selected, the probability of finding better offspring in a large offspring space is small so that the advantage of the clustering tournament selection cannot be properly illustrated.

Therefore, in order to further improve the GP search, other directions, including offspring selection, should be considered.

Although in theory tournament size 3 was shown to have the lowest total loss of program diversity for the quadratic FRD in the clustering tournament selection, the experimental results did not show that tournament size 3 is significantly better than others. This might be because the quadratic FRD will not appear if the clustering tournament selection is applied from the beginning of a GP search.

3.8 Chapter Summary

This chapter used the loss of program diversity, the selection frequency, and the selection probability distribution on four populations with different FRDs to simulate parent selection behaviours in the standard tournament selection, the no-replacement tournament selection, our round-replacement tournament selection and our clustering tournament selection. It also provided experimental analyses of the round-replacement and the clustering tournament selections in three different problem domains. The simulations and experimental analyses provided additional insight into the parent selection pressure in tournament selection and the outcomes are as follows.

- The selection pressure is mainly controlled by tournament size and is aggravated by a skewed FRD in standard tournament selection. Population size seems to have little impact on parent selection pressure under the assumption that the standard breeding process is used. Therefore, when determining tournament size for an intended parent selection pressure, the actual population size need not to be considered.

- This chapter showed that the multi-sampled issue *seldom* occurs in standard tournament selection when common and realistic tournament sizes and population sizes are used. Therefore, although the sampling-without-replacement strategy in no-replacement tournament selection can solve the multi-sampled issue, there is no significantly different selection behaviour between no-replacement and standard tournament selection schemes. The results justify the common use of the simple sampling-with-replacement scheme.

- The not-sampled issue occurs when smaller tournament sizes are used in standard tournament selection. Our round-replacement tournament selection using an alternative sampling-without-replacement strategy can solve the issue without altering other aspects in the standard tournament selection. The different selection behaviour in the round-replacement tournament selection compared with the standard one leads to better GP search results only when tournament size 2 is used for some problems (ones that need low parent selection pressure in order to find acceptable solutions). Overall, solving the not-sampled issue does not appear to significantly improve a GP system: the not-sampled issue in standard tournament selection is not critical.

- Different sampling replacement strategies have little impact on the parent selection pressure. Eliminating the multi-sampled issue and the not-sampled issues did not significantly change the selection behaviour in standard tournament selection and did not tune the selection pressure in dynamic evolution.

- The high between-group selection pressure issue has a strong interaction relationship with the FRD of a population. FRDs change generation by generation and can be seen as the analogue of the dynamics in evolution. Using the knowledge of FRD is a promising way to modify the standard tournament selection in order to tune the parent selection pressure dynamically and automatically. The clustering tournament selection is a such strategy and is worth further investigation. It can significantly improve GP search performance for some problems, although may not be required for easy problems (i.e. EvePar). There are likely to be other, more effective population clustering methods other than merely using the fitness values. Nonetheless, in light of the results presented in Section 3.7, we hope that researchers will be encouraged to experiment with the simple population clustering method in the initial stages of the development of their alternative parent selection algorithms.

As tournament selection requires knowledge of only the fitness rank of an individual and is independent of the representation of the individual [146], we expect that the results of our tournament selection analyses can be applied directly to other forms of EAs.

Chapter 4

Improving Parent Selection Efficiency

The previous chapter analysed parent selection behaviour in tournament selection and provided guidance on how to optimise parent selection *pressure* along the evolutionary process. This chapter investigates ways to improve parent selection *efficiency*. It firstly presents two approaches to improving the efficiency of the standard tournament selection and the clustering tournament selection. It then introduces a framework for gathering information on good predecessor programs and shows, via sets of experiments, that only a small fraction of all evaluated programs contributes to the best program found. It argues that parent selection efficiency may be improved if the good predecessor programs or corresponding correlates can be identified in advance.

4.1 Introduction

As stated in Section 2.5, fitness evaluation cost is an important selection-related open issue. The cost needs to be effectively reduced in order to improve the efficiency of GP search. For a generational tree-based GP, the total number of fitness evaluations in evolution is generally a function of population size, individual size, number of training fitness cases, and number of generations required to find acceptable solutions. Therefore, there are several ways to reduce the fitness evaluation cost.

One is to improve the search efficiency of EAs so that a smaller number of generations are required, leading to overall savings on fitness evaluation. Other ways are to shrink or dynamically change the population size [12, 46, 110, 156, 185] and to control code bloat [147] to reduce the fitness evaluation cost. In some circumstances, approximate fitness values are acceptable. Therefore, fitness estimation [86, 91], which evaluates only individual representatives, or fitness approximation [4, 54, 56, 178], which uses only a portion of the given training fitness cases, are used to reduce the fitness evaluation cost.

Other directions, for instance, fitness caching [62, 199], fitness inheritance [161], fitness evaluation avoidance [84], and backward-chaining evolutionary algorithms [146], focusing on avoiding unnecessary evaluations, are also very interesting. Some detailed reviews can be found in Section 2.5 on page 28.

4.2 Chapter Goals

Since tournament selection, which is used in this thesis, is a fitness-driven parent selection method, this chapter investigates ways to improve the efficiency of tournament selection for parents. In particular, this chapter addresses the following research questions:

- How can the characteristics of the standard tournament selection be exploited to reduce the fitness evaluation cost in the parent selection phase without reducing the effectiveness of the GP search?

- How can the fitness evaluation cost be reduced while taking advantage of the clustering tournament selection?

- Is there any other possible way to *minimise* the fitness evaluation cost in the tournament selection for parents?

4.3 Utilising the Characteristics of Standard Tournament Selection

The loss of program diversity analyses for standard tournament selection in Section 3.4.3 and for round-replacement tournament selection in Section 3.6 showed that the existence of not-sampled individuals is one of the characteristics of the standard tournament selection for smaller tournament sizes. According to that analysis, the not-sampled issue is harmless, and, therefore not-sampled individuals can be utilised to reduce the fitness evaluation cost when smaller tournament sizes are used by avoiding the evaluation of those not-sampled individual programs.

4.3.1 Ejit

We propose a simple algorithm called *Evaluated-just-in-time* (Ejit) and expect it to provide constant savings as long as there exist sufficient not-sampled individual programs. Briefly, Ejit works in the following way:

1. follow the standard procedure to create programs at a generation g but *do not* evaluate them,

2. sample programs at generation g for tournaments,

3. evaluate the sampled programs if they have not been evaluated, then select the winners as the parents of programs at the next generation.

Clearly, in Ejit, individual programs that have not been sampled at all will never be evaluated. The fitness evaluations which used to be processed unnecessarily for those not-sampled individual programs are therefore avoided. According to the sampling probability model for the standard tournament selection in Section 3.4.1 on page 51, the expected computational saving is 36.8%, 13.5%, 5.0%, 1.8%, ,0.7%, 0.25%, and 0.09% for tournament size from 1 to 7, respectively. Clearly, the saving decreases as tournament size increases and becomes very limited when the tournament size is greater than three.

4.3.2 Experiment results

To test the efficiency of Ejit, we repeated the same set of experiments described in Section 3.6.4 using the same sets of GP configuration but with the Ejit algorithm applied. Table 4.1 shows the computational savings obtained by Ejit. Clearly, the experimental results support the expected savings estimated from the mathematical sampling probability model.

Table 4.1: Computational savings on not-sampled individual programs (%).

Tournament Size	EvePar	SymReg	BinCla
2	13.55 ± 0.14	13.57 ± 0.12	13.57 ± 0.14
3	5.01 ± 0.08	5.01 ± 0.09	4.98 ± 0.09
4	1.86 ± 0.06	1.86 ± 0.06	1.85 ± 0.06
7	0.10 ± 0.01	0.09 ± 0.01	0.10 ± 0.01

4.4 Analysis of EMS-EA and BC-EA

After we completed the analysis of Ejit, we found a very closely-related work from a recent publication, the efficient macro-selection EA (EMS-EA) and the backward-chaining EA (BC-EA) algorithms [146]. However, our research was conducted independently.

EMS-EA and BC-EA are closely related to Ejit because they share the same foundation, utilising the characteristic of having not-sampled individuals. Poli and Langdon [146] concluded that BC-EA is better than EMS-EA and its efficiency was demonstrated through sets of experiments using tournament sizes 2 and 3. The following subsections briefly review and analyse the EMS-EA and BC-EA algorithms in order to provide some insight into the strength and weakness of these two algorithms, as well as our Ejit algorithm.

4.4.1 A brief review of EMS-EA and BC-EA

In EMS-EA, the maximum number of generations G has been set, and a sequence of genetic operators that will be used to create the entire population at each generation is determined according to the predefined crossover and mutation rates.

However, the sequence of genetic operators is just memorised at this stage without execution. For each of the individuals that are required for executing each of the memorised genetic operators, according to the predefined tournament size, IDs of a number of parent individuals that need to be sampled into each tournament are also memorised. In other words, tournament selections are *virtually* conducted by just memorising the sampled individual IDs, and genetic operators are also *virtually* applied by generating random offspring IDs and making connections between the sampled individual IDs and the offspring IDs. For instance, if the tournament size is k, at a non-initial generation g, each individual has connections from at most k individuals at the previous generation $g-1$. However, it is not necessary that every individual at generation $g-1$ is connected to individuals at generation g, especially when k is small. At the end, a graph structure that describes the *weak ancestral relationship*[1] between individuals across the whole *virtual* evolutionary process is constructed. Then a post-process on the graph is conducted by starting from the individuals at the last generation and tracing back to the initial generation, and marking individuals that are not involved in tournaments as *neglected*. Finally, the real evolutionary process starts by randomly creating and evaluating the individuals whose IDs are not marked at the initial generation and performing the memorised genetic operators to generate offspring whose ID is also unmarked at the next generation, and so on. Individuals who are marked as neglected will not be created and evaluated, thus computational savings can be obtained. According to the model given in [146], EMS-EA can provide about 13.5%, 5.0%, and 1.8% savings for tournament size 2, 3, and 4 respectively. The algorithm is given in Figure 4.1.

In order to more rapidly find better solutions and possibly to further increase savings, Poli and Langdon then proposed BC-EA. Briefly, BC-EA starts from an individual at the last generation G and uses the depth-first search to determine all *possible* ancestors all the way back to the initial generation. It then creates and evaluates these ancestors and moves forward to the last generation. The process then repeats for another individual at the last generation and so on. The algorithm is given in Figure 4.2.

[1]This is because not every sampled individual can become a parent due to the selection pressure.

```
1: for gen = 1 to G do
2:    for ind = 1 to M do
3:       op[gen][ind] = choose genetic operator
4:       for arg = 1 to arity(op[gen][ind]) do
5:          pool[gen][ind][arg] = choose k random individuals drawing from pop[gen − 1]
6:       end for
7:    end for
8: end for
9: Analyse connected components in pool array and calculate neglected array
10: Randomly initialise individuals in population pop[0] except those
11: marked in neglected[0], calculate fitness values, and store them in vector fit[0]
12: for gen = 1 to G do
13:    for ind = 1 to M do
14:       if not(neglected[gen][ind]) then
15:          for arg = 1 to arity(op[gen][ind]) do
16:             w[arg]=select winner from pool[gen][ind][arg] based on fitness in fit[gen-1]
17:          end for
18:          pop[gen][ind]=result of running operator op[gen][ind] with arguments w[1], ...
19:          fit[gen][ind] = fitness of pop[gen][ind]
20:       end if
21:    end for
22: end for
```

Figure 4.1: EMS-EA from [146]

```
1: Let r be an individual in the population at generation G
2: Choose an operator to apply to generate r
3: Do tournaments to select the parents:
   s_1, s_2, ... = individuals in generation G − 1 involved in the tournaments
4: Do recursion using each unknown s_j as a subgoal. Recursion terminates
   at generation 0 or when the individual is known(i.e. has been evaluated before).
5: Repeat for all individuals of interest at generation G
```

Figure 4.2: BC-EA from [146]

There is a special property in BC-EA. To describe the property in a simpler form, we make a couple of simple assumptions that are consistent with that in [146]: assume a two-offspring crossover operator and a one-offspring mutation operator are used, that N be the population size so that the total number of tournaments for selecting parents is also N, and assume the tournament size $k = 2$. From the algorithm, if only one individual at the last generation G needs to be evaluated, assuming the individual is generated by mutation, then two distinct individuals will need to be sampled and evaluated at generation $G - 1$. If the two individuals at generation $G - 1$ are generated by crossover, then approximately four distinct individuals will need to be sampled and evaluated at gener-

ation $G-2$. Therefore, the approximate number of distinct individuals that will be evaluated in the immediate previous generation grows exponentially until it hits the upper bound or the initial generation is reached, whichever is earlier. A simple upper bound for this case based on the model given in [146] is $\frac{k}{k-1}N$. It will be the same as the population size when $k=2$ and will be greater than the population size when $k>2$. Therefore, it seems that the simple upper bound model given in [146] may be inappropriate. We think the upper bound should be $N(1-(\frac{N-1}{N})^{2N})$ by taking off the expected number of individuals that will never be sampled into any tournament, which is $N(\frac{N-1}{N})^{2N}$ in this case, from the population. We use our upper bound model in the later analysis. The growing period g_e is called the *transient* period. It can be estimated by the following equation:

$$g_e \approx log_k^{N/m} \tag{4.1}$$

where m is the number of individuals evaluated at generation G.

This special property provides BC-EA the other source of saving compared with EMS-EA. Individuals that are not sampled during the transient period make an extra contribution to the saving in BC-EA. The longer the transient period, the larger the extra saving.

The authors concluded that BC-EA can provide around 20% savings in terms of numbers of evaluations than a conventional GP when using $k=2$ and possibly over 35% savings for very low selection pressures. The authors also concluded that BC-EA is superior to EMS-EA due to the following reasons:

- It offers a combination of fast convergence (increased efficiency in terms of fitness evaluations) and complete statistical equivalence to a standard EA.
- It can be fruitfully applied to large tournament size. "For example, with BC-EA, tournament size 7, and a population of a million individuals — which is not unusual in some EAs such as GP — one could calculate 1 individual at generation 7, 7 individuals at generation 6, 49 individuals at generation 5, etc. at a cost inferior to that required to initialise the population in a forward EA."[146]

The authors pointed out that BC-EA is an area worthy of further investigation.

4.4.2 Comparing EMS-EA and BC-EA

4.4.2.1 memory usage and search behaviour

In terms of memory used, EMS-EA and BC-EA both need to memorise the individual IDs involved in tournaments and the sequence of genetic operators. Furthermore, BC-EA has to use extra memory to store all the fitness values of individuals that have been evaluated, which is not necessary for EMS-EA.

In terms of search behaviour, BC-EA appears to differ from EMS-EA. The difference is the order of those memorised components, and consequently the order in which individuals in the population are evaluated. In EMS-EA, individual IDs and genetic operators are memorised generation by generation; thus individuals are evaluated generation by generation. In BC-EA, they are memorised as a result of a recursive depth-first search; thus individuals evaluated are across different generations, moving back and forth. Since fitness values in later generations tend to be better than those in earlier generations, the difference in the order of evaluating individuals led to a claim that BC-EA tends to find better solutions faster than EMS-EA in the early half of a run but slower in the later half [146]. However, finding better solutions faster in the early half of a run does not necessarily mean finding acceptable overall solutions faster. Therefore, the overall value of an algorithm which is able to find better solutions faster in the early half of a run is questionable.

4.4.2.2 computational saving

In terms of computational saving, BC-EA may be able to provide more savings than EMS-EA if the following conditions meet sequentially:

- The decision on setting the last generation G is correct.
- The number of individuals m at the last generation that needs to be evaluated for finding an acceptable solution is small so that there exists a transient period.

However, to properly determine the last generation is not trivial. This is why another stopping criterion, which is the maximum number of generations with-

out improvement (*max-gwi*) has been introduced [51]. Incorrect decisions on the last generation will seriously affect the saving ability of BC-EA and reduce its feasibility. It is quite possible that no acceptable solution can be found by BC-EA after evaluating all individuals at the predefined last generation, whereas the conventional EA may be able to find an acceptable solution in a later generation by using the max-gwi stopping criterion.

The transient period is controlled by the tournament size, the number of tournaments required to select a sufficient number of parents, and most importantly the number of individuals evaluated at the last generation G. The probability of finding the best individual of a population within a small proportion of the population at the last generation is the same as the proportion of the population evaluated. For instance, suppose the last generation G is correctly determined: if all the fitness values are distinct, then the probability of finding the best of the population within the first $n\%$ of the population evaluated is just $n\%$. This probability also depends on the proportion of the population having the same best fitness value. For easy problems, the proportion may be larger, and in such cases, BC-EA may be able to find the best earlier thus provide more saving (though it cannot be sure without evaluating all the population).

Furthermore, although the authors claimed that BC-EA can provide saving even with larger tournament sizes, they assumed a very large population and also assumed that picking a random individual of the last generation would give an acceptable solution. Although the authors provided some arguments, these assumptions are not proven in many cases.

4.4.2.3 missing element

We think that one important comparison is missing in [146]. The authors did not compare the search performance of using the tournament size 2 or 3 with that of using larger tournament sizes, for instance 4 or 7.

Sometimes low parent selection pressure cannot reliably drive the search to find acceptable solutions within a given number of generations whereas high parent selection pressure can. As a result, when using low parent selection pressure, the total number of generations needed to find an acceptable solution can be

much larger than when using high parent selection pressure. Although savings can be obtained from not-sampled individuals at each generation with low parent selection pressure, overall the total number of individuals evaluated can be much larger than the number evaluated when using high parent selection pressure. For this reason, it seems that focusing only on tournament size 2 or 3 to analyse the saving ability of EMS-EA and BC-EA makes their study incomplete.

To investigate whether a variation of an algorithm can provide extra savings, it is necessary to not only compare its efficiency with that of the standard algorithm or other variations using the same set of parameters, but also to ensure it can provide comparable or better problem-solving quality than the standard algorithm or other variations using tuned parameters. It would be inappropriate if the latter has not been taken into account.

4.4.3 Experiment results

In order to further investigate the saving ability of BC-EA, we followed the instructions given in [146] to reproduce the experiments for their two symbolic regression problems, Poly4 and Poly10 (shown in Equations 4.2 and 4.3). We constructed two conventional GP systems using tournament sizes 2 and 7 respectively. Four different population sizes (100, 1000, 10000 and 100000) were used for both Poly4 and Poly10 in [146]. From the four population sizes, we arbitrarily picked a population size 1000 for Poly4 and a population size 10,000 for Poly10. We set the maximum number of generations to 50 and conducted 100 independent runs. The fitness function, the function set, the crossover rate, and the mutation rate are the same as those in [146]. We also use the same two-offspring subtree crossover with uniform random selection of crossover points as in [146]. However, we have not implemented the point mutation operator used in [146]. Our mutation operator is the conventional subtree mutation [93].

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_3x_4 + x_1x_4 \tag{4.2}$$

$$f(x_1, x_2, ..., x_{10}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10} \tag{4.3}$$

Table 4.2 shows the performance of using tournament size 2 and 7. For instance, when tournament size 2 is used and the maximum number of generations G is 50, 42 out of 100 runs can find the optimal solution for Poly4. The average number of generations processed to find the optimal solution over the 42 runs is 31 (the standard deviation is 11). The average fitness value over the 100 runs is 4.6 (the standard deviation is 4.2). When tournament size 7 is used and G is 50, 83 out of 100 runs can find the optimal solution. The average number of generations required to find the optimal solution over the 83 runs is 13. The average fitness value over the 100 runs is 1.3.

Table 4.2: Performance comparison between tournament sizes 2 and 7 for poly4 and poly10 problems.

Problem	k	G	# of Success	Generations Required for Success	Sum of Error
poly4	2	50	42	31 ± 11	4.6 ± 4.2
		100	74	47 ± 22	1.6 ± 2.9
	7	50	83	13 ± 11	1.3 ± 3.0
poly10	2	50	0	–	5.4 ± 1.2
		100	7	69 ± 15	4.6 ± 1.8
	7	50	11	36 ± 7	4.4 ± 1.9

From the table, it is clear that by using tournament size 7, for both Poly4 and Poly10, more runs can find the optimal solutions, significantly fewer generations are required on average, and the fitness value is better on average. Due to the incomparable problem-solving qualities, it would be pointless to compare the saving ability between the use of two different tournament sizes. Therefore, we conducted additional sets of experiments by gradually increasing G for the GP system using tournament size 2 with the aim of making its average fitness value comparable with that in the GP system using tournament size 7. When the maximum number of generations reached 100, the tournament size two systems were approaching the average fitness values of the tournament size seven systems. The corresponding performance is also reported in Table 4.2.

4.4.4 Efficiency analysis

Table 4.3 shows the estimated total number of evaluations conducted in the conventional GP, GP with EMS (EMS-GP), and GP with backward-chaining (BC-GP) for Poly4 and Poly10 problems. Note that we ignored the cases for $G = 50$ when $k = 2$ as they did not provide comparable problem-solving qualities.

Table 4.3: Efficiency comparison between conventional GP, EMS-GP and BC-GP using tournament sizes 2 and 7 for Poly4 and Poly10 problems. Note that the total number of evaluations for BC-GP is estimated according to the best assumptions.

Problem	k	G	Total # of Evaluations ($\times 10^6$)		
			Conventional GP	EMS-GP	BC-GP
poly4	2	100	6.1	5.3	4.7
	7	50	1.9	1.9	1.6
poly10	2	100	97.8	84.6	83.9
	7	50	48.5	48.4	47.9

For Poly4 with $k = 2$ and $G = 100$, the total number of individuals evaluated in the conventional GP system is approximately $(74 \times 47 + 26 \times 100) \times 1000 \approx 6.1 \times 10^6$. The total number of individuals evaluated in a GP system with EMS under the same conditions can be easily estimated as $(74 \times 47 + 26 \times 100) \times 1000 \times (1 - 13.5\%) \approx 5.3 \times 10^6$. Although we have not yet implemented a backward-chaining GP system, we can make an assumption that the optimal solution is the first individual evaluated at the last generation. Such an assumption is the best scenario for a backward-chaining GP system. Under this assumption, we estimated the corresponding total number of individuals evaluated in the following steps:

- calculate the transient period, which is approximately $log_2^{1000\times(1-13.5\%)} = 9.75 \approx 10$
- calculate the sum of evaluated individuals within the transient period, which is approximately $\frac{1\times(1-2^{10})}{1-2} = 1023$
- calculate the total number of evaluated individuals in the whole evolutionary process, which is approximately $74 \times 1023 + (74 \times (47 - 10) + 26 \times 100) \times 1000 \times (1 - 13.5\%) \approx 4.7 \times 10^6$

For Poly4 with $k = 7$ and $G = 50$, the total number of individuals evaluated in the conventional GP system is approximately $(83 \times 13 + 17 \times 50) \times 1000 \approx 1.9 \times 10^6$.

From the estimated results above, although a backward-chaining GP system (BC-GP) in the best scenario can evaluate a smaller number of individuals than a GP with EMS (EMS-GP) and the conventional GP when tournament size 2 is used, it needs to evaluate just over twice as many individuals than the conventional GP using tournament size 7 in order to provide the similar problem-solving quality. A similar finding for Poly10 is obtained.

We now compared the efficiency between the conventional GP, a GP with EMS and a backward-chaining GP using the tournament size 7. Based on the model given in [146], for tournament size 7 only about 0.09% of population will not be sampled, a GP with EMS will not be able to provide a large saving and will be effectively the same as the conventional GP, whereas a backward-chaining GP may be able to provide some saving from the not-sampled individuals in the transient period. Let us again assume the optimal solution is the first individual evaluated at the last generation for BC-EA. For Poly4, the transient period is approximately $log_7^{1000\times(1-0.09\%)} = 3.55 \approx 4$ generations. Within the *last four* generations, a backward-chaining GP needs to only evaluate approximately $\frac{1+7+49+343}{4\times1000} = 10\%$ of the total number of individuals that need to be evaluated in the conventional GP, obtaining 90% saving in the transient period. For Poly10, the transient period is approximately 5 generations. Within the *last five* generations, a backward-chaining GP needs to only evaluate approximately 6% of the total number of individuals that need to be evaluated in the conventional GP, obtaining 94% saving in the transient period. These results seem to support the claim given in [146]. However, the overall savings in whole evolutionary processes obtained by backward-chaining GP compared to the conventional GP are only $1 - (1.6/1.9)100\% \approx 16\%$ and $1 - (47.9/48.5) \approx 1\%$ for Poly4 and Poly10, respectively. Furthermore, note that there are two important assumptions: the last generation is correctly determined and the optimal solution is the first individual evaluated at the last generation. Since neither assumption is likely to hold, the saving ability of BC-EA is much less than claimed in [146].

4.4.5 Limitations of EMS-EA and BC-EA

The ability of EMS-EA to provide computational savings is limited by the tournament size. More precisely, saving can be obtained only when smaller tournament sizes are used. For problems that can easily be solved under high selection pressure, there will be no clear saving.

The ability of BC-EA to provide computational savings is limited by three primitive factors and one derived factor. The three primitive factors are the tournament size, the decision on the last generation, and the number of individuals that need to be evaluated at the last generation. The derived factor is the length of the transient period. The transient period will become shorter if more individuals at the last generation will be evaluated, and will not exist if the full population of the last generation will be evaluated. In summary, the limitations of BC-EA include:

- If one wants to use tournaments of sizes more than three and to compute a large proportion of the final generation, the computation saving provided by BC-EA may be too limited[2] and dependent on the ratio of the transient period to the total number of generations used.

- The decisions on the last generation G and the number of individuals m that should be evaluated at the last generation directly affect the ability of BC-EA to provide computational savings. Making good decisions is very difficult and becomes the bottleneck of applying BC-EA to real world hard problems.

4.5 Comparing Ejit with EMS-EA and BC-EA

As presented in the previous sections, Ejit algorithm shares the same foundation as EMS-EA and BC-EA and is very close to EMS-EA. It is natural to expect Ejit and EMS-EA to provide a *similar* amount of savings in terms of the number of individuals evaluated. EMS-EA will not create and evaluate a program p if all

[2]This was clearly stated in [146] by the authors themselves.

of its *direct weak descendants* are not sampled. Direct weak descendants are defined as offspring which are produced by winners of the tournaments in which the program p participated but is not necessarily the winner. Ejit does not make this check. However, this difference leads only to a negligible difference in savings between EMS-EA and Ejit because the probability that all of the direct weak descendants of the program p are not sampled is very small, even for tournament size 2.

If L is the event that all of the direct weak descendants of the program p are not sampled, the probability of L is:

$$P(L) = \left(\frac{N-n}{N}\right)^{kN} \tag{4.4}$$

where N is the population size, k is the tournament size, and n is the expected number of all direct weak descendants. To calculate n, we need the probability that p is sampled in a single tournament, which is $(1-(\frac{N-1}{N})^k)$. We then multiply the probability by the total number of tournaments N so that $n = N(1-(\frac{N-1}{N})^k)$. For instance, when $k = 2$ and $N = 1000$, $P(L)$ is about 1.8%. When $k = 3$, $P(L)$ becomes 0.01%.

It is also natural to expect Ejit and EMS-EA to have the same set of limitations. However, the Ejit algorithm does have its own features.

When comparing with EMS-EA, Ejit does not require any additional memory to store the sequences of program IDs and genetic operators from the whole evolutionary process. Consequently, Ejit does not require the post-process to identify what individual programs will not be sampled. In other words, Ejit does not spend any overhead on the pre- and post-processes. The decision on which program should be evaluated comes up naturally (hence the name of the algorithm).

When comparing with BC-EA, in addition to not requiring any additional memory, Ejit does not need to choose an appropriate G in order to solve a problem: it can work well with the max-gwi strategy. Ejit can provide constant savings at every generation, regardless of the number of generations used. Its efficiency will never be affected by the length of the transient period and the number of individuals that need to be evaluated at the last generation G.

Although Ejit has to create every individual in a population, including not-sampled individuals which will not be created by EMS-EA and BC-EA, the creation time is very short and the cost of creating not-sampled individuals is in fact negligible.

Like EMS-EA and BC-EA, Ejit gives significant savings only for the cases in which low selection pressure is better for solving given problems and only works for the standard tournament selection. In the previous chapter, the clustering tournament selection was demonstrated to be a promising research direction to improve the standard tournament selection. Ejit, as well as EMS-EA and BC-EA, will not be able to work with the clustering tournament selection since a population needs to be clustered based on fitness values, which requires the population to be fully evaluated beforehand, not be evaluated just in time. In the next section we will investigate other strategies to reduce the fitness evaluation cost in the parent selection phase which will not have the limitation on tournament size and can take advantage of the clustering tournament selection.

4.6 Population Clustering

This section presents a simple but novel population clustering algorithm for GP in order to reduce the fitness evaluation cost while taking advantage of the clustering tournament selection.

Briefly, we cluster the whole population by a heuristic, and select a cluster representative for each cluster. The fitness value of the representative is calculated on all training cases and then directly assigned to other members in the same cluster in order to reduce the fitness evaluation cost.

The central idea of our approach is based on the observation that two programs that are equivalent (in the sense that they compute the same function of their inputs) must necessarily have the same fitness value. If we could identify clusters of equivalent programs, then it would be necessary only to evaluate the fitness of one program in each cluster, and use the same fitness value for all the other programs in the cluster, avoiding the cost of evaluating the fitness of the other programs in the cluster.

In fact, it is adequate to put programs into a cluster if these programs compute the same output values on all the given training fitness cases, regardless of their output values on other inputs, since the fitness of a program depends only on its outputs on the given fitness cases. We consider such programs "*fitness-case-equivalent*". *"Fitness-case-equivalence"* is actually more useful than true equivalence since the clusters may be larger, and therefore generate greater saving.

The problem with this idea is that the obvious way of determining fitness-case-equivalence requires evaluating all the programs on all the fitness cases, which is the time-consuming computation that we are trying to avoid. Instead, we use a heuristic estimate of fitness-case-equivalence based on evaluating programs on a small number of the training fitness cases. This is done by making the heuristic assumption that programs that generate the same output values on a small random set of the training fitness cases are likely to be equivalent on all training fitness cases. The determination of the clusters is woven into the final fitness evaluation so that no repeated fitness case evaluations are performed.

Once the fitness of each cluster has been computed, the clustering tournament selection can be used straight away.

The rest of this section describes the details of the heuristic population clustering algorithm, followed by experiments and analyses.

4.6.1 Heuristic estimate of fitness-case-equivalence

At each generation during the evolutionary process, the algorithm starts by treating the entire population as a single cluster. Then it feeds the first training case into the programs and partitions the cluster into new clusters based on the program outputs. For each newly-formed cluster, the partitioning process is applied again with the next fitness case until no new cluster is formed. The algorithm currently assumes that it has seen enough training cases to determine a cluster once all the programs in a cluster have the same output in two successive training cases. To reduce the chance of premature stopping where a cluster contains non-fitness-case-equivalent programs, the algorithm presents the training cases in a different random order in each generation.

The population clustering algorithm is outlined in Figure 4.3 and illustrated below using a simple one-variable symbolic regression example.

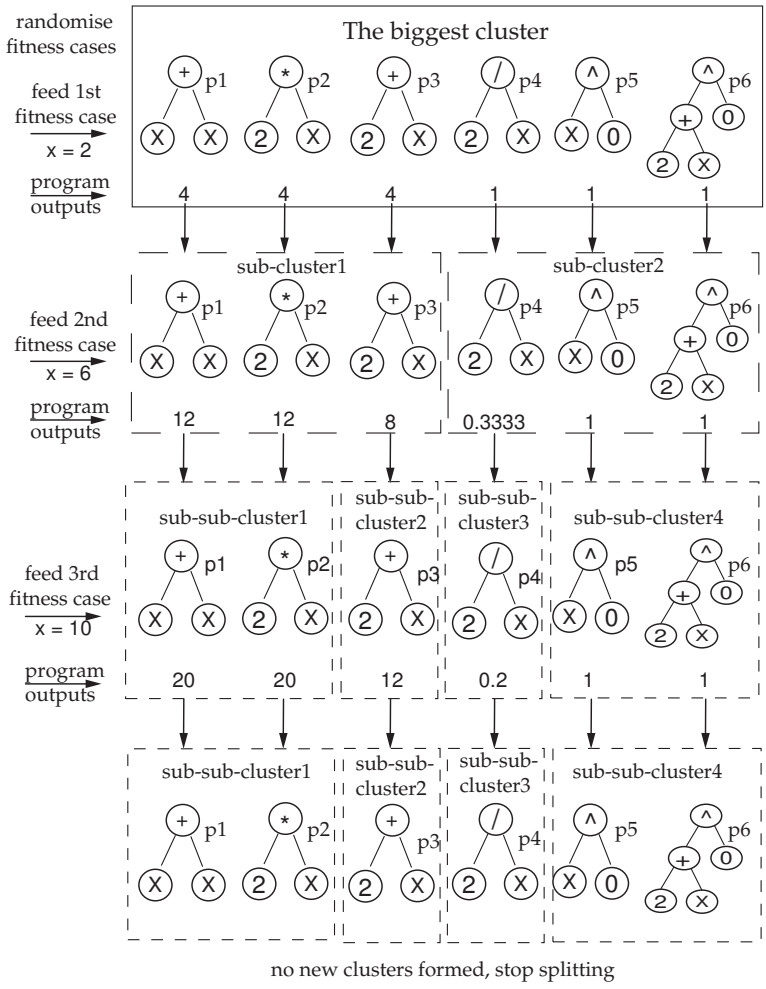


Figure 4.3: Population clustering algorithm.

- *initialisation:* Treat the initial population consisting of 6 programs as one big cluster and randomise the order of the fitness cases.

- *Iteration 1:* Feed the first fitness case `x=2` to each program. The program outputs are `4`, `4`, `4`, `1`, `1`, and `1` respectively, which leaves us with two sub-clusters, one with the program output of `4` and the other with `1`. The initial cluster is replaced by the two sub-clusters.

- *Iteration 2:* Feed the second fitness case `x=6` to the programs in each sub-cluster. The outputs of programs in sub-cluster1 are `12`, `12`, and `8`. Therefore, the sub-cluster is further partitioned into two sub-sub-clusters, one with programs with output value `12`, the other holding a program with an output value of `8`. Similarly, sub-cluster2 is partitioned into two new sub-sub-clusters. Now we have four clusters (sub-sub-clusters1, 2, 3, 4).

- *Iteration 3:* Feed the third fitness case `x=10` to programs in each sub-sub-cluster. According to the program outputs, the same set of clusters remains. As no new cluster is formed, the partitioning process completes.

4.6.2 Fitness evaluation and assignment

Upon completing the population clustering, we progress to the fitness evaluation stage. For each cluster, the program with the least program complexity is chosen as the cluster representative. In this study, the number of nodes is used as a proxy for program complexity; that is, the program with the smallest number of nodes will be selected as a representative for a given cluster. The fitness value of the cluster representative is calculated from the result of evaluating the program on all the training cases (cases evaluated during the clustering stage are not re-evaluated). As all members in a cluster are assumed to be fitness-case-equivalent, the fitness of each cluster representative is directly assigned to the cluster and to all the other members of the cluster.

4.6.3 Experimental results and analysis

To test the effectiveness of the heuristic fitness-case-equivalence population clustering algorithm and its impact on the clustering tournament selection, we repeated the same set of experiments described in Section 3.6.4 on page 76 using a GP system in which the population is clustered using the Heuristic fitness-case-equivalence population clustering algorithm and the parent selection method is the Clustering tournament selection (HCGP).

The experimental results varied on the three different problems. The next subsections analyse the performance of HCGP and compare it with two other GP systems. One is the Standard GP system in which the population is not clustered and the parent selection is the standard tournament selection (SGP). The other is a GP system in which the population is clustered based on Fitness and the parent selection is the Clustering tournament selection (FCGP).

We used three measures for analysing the computational saving, representing three levels of precision. The first is at the coarse level. It is the minimum number

of generations required to find the best-of-run in a run. The second is at a finer level. It is the total number of individual programs that have been evaluated using all given training fitness cases. The third is at the finest level. It is the total number of fitness case evaluations in a run (whether for clustering a population or for calculating the fitness value).

In SGP and FCGP, the second measure is the product of the minimum number of generations and the population size and the third measure is the product of the total number of fully-evaluated programs and the size of the training data set. Since the population size and the size of the training data set are fixed during evolution in general, the three measures are effectively the same for SGP and FCGP. However, in HCGP, the second measure can indicate how often programs are fitness-case-equivalent and how many clusters will be formed. The third measure can show the actual savings on the fitness evaluation. Furthermore, the third measure can be used together with the other two measures to determine the average number of training fitness cases needed to cluster a population, indicating how effective the heuristic fitness-case-equivalence population clustering algorithm is.

4.6.3.1 GCGP and EvePar

The initial results for EvePar show that HCGP does not work well. No run could find an optimal solution. The average misses are 30, 29, and 31 for tournament sizes 2, 4, and 7 respectively. The effectiveness of HCGP is much worse than that of SGP. Therefore, although the total number of fitness case evaluations is smaller, it is not necessary to conduct a further comparison in terms of saving.

For EvePar, the low effectiveness in HCGP may be because two different outputs, either *true* or *false*, are not sufficient to distinguish two different programs. This may also indicate that HCGP may not work for other Boolean problems.

To overcome the issue, we developed another population clustering algorithm for EvePar. It clusters a population according to the program *genotype*[3], grouping syntactically-identical programs. After that, any program in a cluster can be treated as the clustering representative as all programs in the cluster are exactly

[3]It is used according to the term *genotype diversity* introduced in [60, 98].

the same.

We then repeated the set of experiments for EvePar using a GP system with the Genotype population clustering and the Clustering tournament selection (GCGP). Table 4.4 illustrates the effectiveness of GCGP. Tables 4.5 and 4.6 show the saving ability of GCGP. Note that the third measure is omitted. This is because in GCGP it is not necessary to feed any fitness cases to programs for clustering a population, thus the second and the third measures are effectively the same.

Table 4.4: Failure rates (%)for EvePar (Some results for SGP and FCGP are repeated from Table 3.4 on page 93).

Tournament Size	SGP	FCGP	GCGP
2	100	91.4	99.8
4	80.6	88.0	79.2
7	82.4	88.8	73.6

Table 4.5: Average number of minimum generations required for finding the best-of-run for EvePar. The standard deviation follows the $\pm$ sign.

Tournament Size	SGP	FCGP	GCGP
2	76.7 ± 19.7	63.9 ± 25.6	76.5 ± 19.8
4	70.2 ± 21.5	60.6 ± 26.2	72.2 ± 20.8
7	62.6 ± 24.5	59.3 ± 26.1	61.6 ± 22.7

Table 4.6: Average total number of fully-evaluated individual programs (10^3) for EvePar. The standard deviation follows the $\pm$ sign.

Tournament Size	SGP	FCGP	GCGP
2	38.7 ± 9.9	32.2 ± 12.9	37.7 ± 9.8
4	35.4 ± 10.8	30.6 ± 13.2	35.4 ± 10.2
7	31.5 ± 12.4	29.9 ± 13.1	29.9 ± 11.0

For SGP, the best problem-solving quality (80.6% failure rate) is given by tournament size 4. For GCGP, the best problem-solving quality (73.6% failure rate) is given by tournament size 7, which is significantly better than SGP using tournament size 4 as the confidence interval at 95% level is (-12.19, -1.81). The advantage of the clustering tournament selection has been further demonstrated on this problem.

Since the best problem-solving quality in FCGP is not better than that in SGP, the following analyses on the computational saving will only focus on SGP and GCGP.

Comparing the total number of fully-evaluated individual programs (or the total number of fitness cases evaluations) between best cases for SGP and GCGP, the saving is $1-(29.9/35.4)\times 100\% = 15.5\%$. However, this is not a fair comparison because GCGP achieved much better problem-solving quality. To make the test fairer, we reduced the number of generations for GCGP until its average problem-solving quality matched that of SGP (72 generations). The performance savings were then increased to 30% [4].

The substantial saving results from two sources. One is the effectiveness of the clustering tournament selection, which shortens the search time. The other is the genotype population clustering algorithm, which avoids the fitness evaluation on identical programs. To determine how much of the savings are contributed by the genotype population clustering algorithm, we calculated the average number of fully-evaluated individual programs at each generation for GCGP with tournament size 7: $\frac{24.7\times 10^3}{50.8} \approx 486$. This result also represents the average number of clusters per generation. Compared with 504 individual programs evaluated at each generation in SGP, the saving in terms of the number of fully-evaluated individual programs per generation in GCGP is 3.6%. Although this amount of saving is small, it is considerably greater than the 0.10% savings due to not-sampled individuals when tournament size is 7 as used by Ejit. This result, together with the significantly better problem-solving quality, further supports the claim that the clustering tournament selection can maintain the population diversity and improve the search performance.

The efficiency of the genotype population clustering algorithm is highly dependent on the probability of having syntactically-identical programs. For problems that require floating point numbers in solutions, the probability of having lots of syntactically-identical programs will be very small, thus the genotype population clustering algorithm may not be able to provide noticeable savings. Although its apparent efficiency could be "improved" by increasing the population size, decreasing the program size, and tuning other parameters to deliberately

[4] With the maximum number of generations set to 72, GCGP with tournament size 7 provides 80.2% failure rate. The average number of minimum generations required for finding the best-of-run is 50.8 and the average total number of fully-evaluated individual programs is 24.7×10^3. Therefore, the saving given by GCGP is about $1 - (24.7/35.4) \times 100\% = 30.2\%$.

produce lots of syntactically-identical programs, such "fiddling" would represent an improper evaluation.

4.6.3.2 HCGP and SymReg

Table 4.7 illustrates the effectiveness of HCGP on SymReg and Tables 4.8, 4.9, and 4.10 show the saving ability of HCGP in terms of the three measures for SymReg.

Table 4.7: Fitness (RMS error) for SymReg (some results for SGP and FCGP are repeated from Table 3.4 on page 93).

Tournament Size	SGP	FCGP	HCGP
2	48.2 ± 5.2	47.6 ± 5.9	46.8 ± 5.7
4	37.6 ± 8.3	36.8 ± 7.9	36.6 ± 7.9
7	40.9 ± 11.3	33.5 ± 8.3	34.0 ± 7.9

Table 4.8: Average number of minimum generations required for finding the best-of-run for SymReg. The standard deviation follows the ± sign.

Tournament Size	SGP	FCGP	HCGP
2	91.6 ± 10.8	88.9 ± 14.0	90.1 ± 12.8
4	91.9 ± 12.1	91.7 ± 10.6	91.9 ± 10.9
7	88.4 ± 19.6	92.8 ± 9.8	91.3 ± 11.7

Table 4.9: Average total number of fully-evaluated individual programs (10^3). The standard deviation follows the ± sign.

Tournament Size	SGP	FCGP	HCGP
2	46.2 ± 5.4	44.8 ± 7.0	43.0 ± 6.2
4	46.3 ± 6.1	46.2 ± 5.3	43.1 ± 5.3
7	44.6 ± 9.9	46.8 ± 4.9	42.0 ± 5.6

Table 4.10: Average total number of fitness case evaluations (10^6) for SymReg.

Tournament Size	SGP	FCGP	HCGP
2	4.62 ± 0.54	4.48 ± 0.70	4.31 ± 0.62
4	4.63 ± 0.61	4.62 ± 0.53	4.32 ± 0.53
7	4.46 ± 0.99	4.68 ± 0.49	4.21 ± 0.56

The effectiveness of HCGP is very close to that of FCGP (the difference is insignificant) and is better than that of SGP for all three different tournament sizes.

The results show that the heuristic fitness-case-equivalence population clustering algorithm can cluster populations as accurately as using the fitness values for this problem.

Since HCGP and FCGP have very similar best problem-solving qualities, we can compare the total number of fitness case evaluations to show that HCGP can provide $1 - (4.21/4.68) \times 100\% \approx 10.0\%$ saving over FCGP. In order to calculate the savings more precisely, we reduced the maximum number of generations for FCGP until its average fitness matched that of HCGP (96 generations). The saving by HCGP was then reduced but still over 7.0%.

Based on the average minimum number of generations, the average number of fully-evaluated programs, the population size, and the size of the training data set, we calculated that the average number of training fitness cases fed to every program in order to cluster a population is only 2.43. As the problem-solving qualities of HCGP and FCGP are very similar, the result shows that the heuristic fitness-case-equivalence population clustering algorithm is very effective for this problem.

4.6.3.3 HCGP and BinCla

Table 4.11 illustrates the effectiveness of HCGP on BinCla and Tables 4.12, 4.13, and 4.14 show the saving ability of HCGP in terms of the three measures for BinCla.

The effectiveness of HCGP is better than that of SGP but worse than that of FCGP for all three different tournament sizes. Although it appears that HCGP can provide 12% savings over FCGP from Table 4.14, the comparison is unfair. After reducing the maximum number of generations for FCGP until its average fitness matched that of HCGP, HCGP did not provide any savings over FCGP.

A possible explanation for HCGP not working very well for BinCla is as follows.

The heuristic estimate of fitness-case-equivalence algorithm works well and has been demonstrated in SymReg. However, the ability to find fitness-case-equivalence efficiently does not provide any advantage for BinCla. This is because deciding which class a given fitness case belongs to is determined only by

the sign of the program output (positive or negative) rather than the actual program output. Therefore, HCGP produced too many small clusters for BinCla, most having only one member. In SGP, the population can be viewed as 504 clusters with one member in each cluster, whilst in FCGP, the population can have up to $569/3 \approx 189$ clusters. In HCGP, the average number of clusters is 436, close to the population size. This may explain why HCGP is only slightly better than SGP but worse than FCGP. The results suggest that a fuzzier estimate of fitness-case-equivalence should be used for BinCla.

Table 4.11: Fitness (error rate %) for BinCla (some results for SGP and FCGP are repeated from Table 3.4 on page 93).

Tournament Size	SGP	FCGP	HCGP
2	9.2 ± 2.9	7.4 ± 2.3	8.4 ± 2.6
4	8.7 ± 2.7	7.7 ± 2.5	8.4 ± 2.5
7	8.7 ± 2.7	7.9 ± 2.5	8.3 ± 2.6

Table 4.12: Average number of minimum generations required for finding the best-of-run for BinCla. The standard deviation follows the $\pm$ sign.

Tournament Size	SGP	FCGP	HCGP
2	57.6 ± 29.0	42.7 ± 29.3	52.5 ± 27.5
4	46.6 ± 29.4	37.1 ± 28.4	42.4 ± 27.3
7	40.2 ± 29.2	35.0 ± 27.8	35.1 ± 26.1

Table 4.13: Average total number of fully-evaluated individual programs (10^3) for BinCla. The standard deviation follows the $\pm$ sign.

Tournament Size	SGP	FCGP	HCGP
2	29.0 ± 14.6	21.5 ± 14.8	24.6 ± 13.2
4	23.5 ± 14.8	18.7 ± 14.3	19.4 ± 12.8
7	20.3 ± 14.7	17.7 ± 14.0	15.5 ± 11.8

Table 4.14: Average total number of fitness case evaluations (10^6) for BinCla. The standard deviation follows the $\pm$ sign.

Tournament Size	SGP	FCGP	HCGP
2	5.51 ± 2.77	4.08 ± 2.80	4.68 ± 2.51
4	4.45 ± 2.80	3.55 ± 2.72	3.69 ± 2.44
7	3.84 ± 2.79	3.35 ± 2.66	2.95 ± 2.25

The analysis results show that HCGP has some saving potential but has no clear advantage over FCGP, and that GCGP worked well for EvePar but may not work well for problems that require real numbers presented in solutions. Developing a more robust and real number-tolerant fuzzy population clustering algorithm together with the clustering tournament selection is a promising direction for improving the efficiency of parent selection and is worth further investigation.

The results in the previous sections have shown that Ejit and the population clustering algorithms have advantages for improving parent selection efficiency but also limitations. It is possible to further improve the parent selection efficiency.

4.7 Using GPPs to Increase Efficiency

Ejit, as well as EMS-EA and BC-EA, have explored ways of reducing fitness evaluations by identifying programs that are not sampled. Not-sampled programs certainly do not contribute to the best-of-run program but the fraction of not-sampled programs is small. It is likely that there are more programs that do not contribute to the best-of-run programs. The section defines programs that are ancestors of the best program found as *Good Predecessor Programs* (GPPs), and hypothesises that there is a small fraction of programs in evolution that are GPPs. Since only GPPs are worthy of evaluating, if the hypothesis is true and if GPPs could be identified in advance, then the cost of fitness evaluation would be minimised. This section does *not* propose a method for identifying GPPs but rather presents some experiments to analyse the feasibility of using GPPs to increase parent selection efficiency.

This section first develops a framework to locate GPPs and gather sufficient information of GPPs from the evolutionary process, then analyses the output of the framework.

4.7.1 The framework

The *GPP set* of a single GP run is the collection of all GPPs of the best program generated in a GP run. It consists of all programs in each generation that are

ancestors (according to the genetic operators of crossover, mutation, and reproduction) of the program with the best fitness value.

The framework constructs the GPP set by recording program ancestry during evolution, and then tracing the best program found in a GP run all the way back to the initial population. The resulting GPP set is then analysed to extract high level important information in order to answer research questions. Figure 4.4 illustrates the relationships among the three components of the framework.

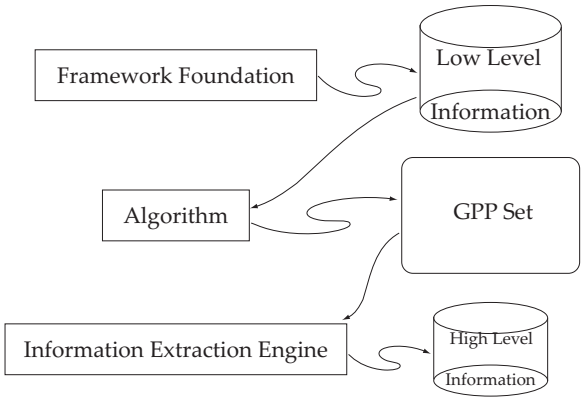


Figure 4.4: The structure of the framework.

A comprehensive log system is used as the framework foundation to record all necessary low level information into a detailed program log file. Each entry in the log file contains the following information which can be used to provide evidence for the analysis:

- program ID
- the generation in which it was created
- how the program was created/generated (new, crossover, mutation, or reproduction)
- IDs of its parents (if any)
- its size (number of nodes)
- its fitness value
- the program as a LISP expression

Table 4.15 shows a few example records of a detailed program log file. The following is a brief interpretation focusing only on how programs are generated.

Table 4.15: Sample records in a detailed program log file.

ID	Gen	How	Parents	Size	Fitness	Program
1	0	new	-1:-1	4	34.75	If(x,3.60,x)
2	0	new	-1:-1	7	37.20	If(Sin(x),x,Mul(x,x))
⋮	⋮	⋮	⋮	⋮	⋮	⋮
105	1	repd	2:-1	7	37.20	If(Sin(x),x,Mul(x,x))
106	1	xovr	1:76	6	28.57	If(x,Add(1.74, x), x)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
218	2	muta	105:-1	8	39.74	If(Sin(x),Abs(x),Mul(x,x)))

Other information can be easily understood. Programs with IDs 1 and 2 at the initial generation were randomly created (and therefore had no parents). Program 105 was generated in the 1st generation by reproducing Program 2 from the initial generation (and therefore has only one parent). Program 106 was generated by applying crossover to Programs 1 and 76 from the initial generation. Program 218 at 2nd generation was generated by mutating Program 105 from the 1st generation.

The log system also keeps track of the ID of the best program found along the evolutionary process. The best program can appear at any generation from the initial to the last. Where there is more than one program with the same fitness value, the system records only the first one found since this will be in the earliest generation. For simplicity the best program is selected only on the basis of its fitness value.

Once a run is completed, the algorithm constructs the GPP set by a depth first search through the log file, starting at the record of the best-of-run program and following links to parent programs, adding all the programs it finds to the GPP set.

4.7.2 High level information extraction

While the GPP set supports the extraction of much more high level information, in this study, we extracted only the number of GPPs at each generation in order to identify the fraction of programs that are directly involved in producing the best program.

Figure 4.5 shows the number of GPPs across all generations in a sample run in our experiments. The sample run was configured with a maximum generation of 200 and a population size of 200. The x-axis shows the generation number and the y-axis shows the fraction of GPPs. In this run, the best program was found in generation 196. (Note that the evolution process was not terminated until generation 200, but no improvement was found in the last four generations.)

In the initial generation, the number of GPPs of the best program is roughly a quarter (26%) of the population. After a little fluctuation, the fraction of GPPs quickly climbed up to a peak of almost a half (47%) in generation 35. The fraction constantly fluctuates during evolution, but tends to shrink towards the end of the evolution.

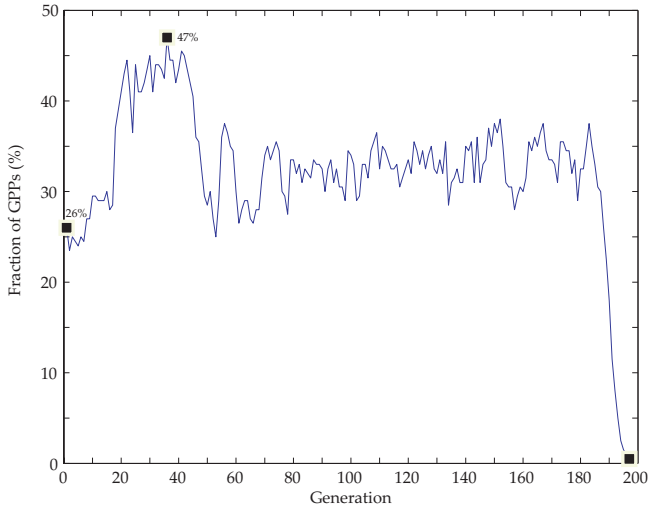


Figure 4.5: Fraction of GPPs in a sample run.

In this sample run, over 50% of the programs at each generation did not contribute to the final best program, and therefore evaluating their fitness was "wasted". This suggests that there is considerable opportunity for reducing the cost of fitness evaluation if we could identify these non GPPs. Of course, a single run is not necessarily indicative of typical behaviour, and the next section describes further experiments on a range of problems.

4.7.3 Experiment design and configuration

Obtaining a robust measure of the fraction of GPPs in a population needs a range of quite different GP scenarios. The experiments used a symbolic regression

problem, a binary classification problem, and a multi-class classification problem from different domains and with increasing levels of difficulty (low, medium and high). It is also necessary to identify the effect of the GP parameters on the fraction of GPPs. There are many possible parameters that could be investigated. The experiments focused on two parameters — tournament size and population size — because they are more likely to influence the fraction of GPPs.

The experiments covered four different tournament sizes — 20, 10, 4, and 1, and six different population sizes — 100, 200, 500, 1000, 2000, 5000. Note that tournament size 1 is equivalent to the random selection, meaning no selection pressure.

4.7.3.1 data sets

The symbolic regression problem (Regression) is shown in equation (1). The experiments generated 100 fitness cases by assigning equally-spaced real numbers in (-10,10] to x.

$$f(x) = \begin{cases} x^2 - x & , x \geq 0 \\ sin(x) + \frac{1}{x} & , x < 0 \end{cases} \tag{4.5}$$

The binary classification problem involves determining whether examples represent a normal liver or a liver disorder. The dataset is the BUPA Liver Disorders dataset (BUPA) chosen from the UCI Machine Learning repository [131]. BUPA consists of 345 data examples, each described by six numeric features.

The multi-class classification problem involves classifying four types of vehicles: *opel, saab, bus* and *van*. The dataset is called Vehicle Silhouette (Vehicle), which was also chosen from the UCI Machine Learning repository. Vehicle consists of 846 data examples, each described by 18 numeric features.

For each classification problem, the data set is split randomly and equally into a training data set, a validation data set, and a test data set with class labellings being evenly distributed across the three data sets for each individual GP run.

4.7.3.2 function set

The function set for the three problems is listed below (see page 78 for detailed explanations)

$$\text{Function Set} \quad = \{+, -, *, /, if, abs, sqrt, sin\} \tag{4.6}$$

The $+$, $-$, and $*$ operators have their usual meanings — addition, subtraction, and multiplication. The / operator represents "protected" division which is the usual division operator except that a division by zero gives a result of zero. Each of these four functions takes two arguments. The *if* function takes three arguments: if the first argument is positive, the *if* function returns its second argument; otherwise, it returns its third argument. The remaining unary functions also have their usual meanings. Note that zero will be returned if the *sqrt* function encounters an invalid argument.

4.7.3.3 terminal sets

There are three terminal sets, one for each problem, with a different number of variables or features in each set. The terminal set for Regression includes a single variable x. The terminal set used in BUPA includes six numerical features. The terminal set used in Vehicle includes 18 numerical features extracted from images of the types of vehicles. More details about those features can be found in [131]. Each terminal set also includes real valued random constants in the range [-5.0, 5.0].

4.7.3.4 fitness function

The fitness function in SymReg is the root-mean-square (RMS) error of the outputs of a program relative to the expected outputs. The fitness function for the classification problems is the classification error rate on the training data set. All problems have an ideal fitness of zero.

For the liver disorder binary classification problem, if the output of a program on a fitness case is negative, the program classifies the fitness case as *disorder*;

otherwise, as *normal*.

The vehicle multi-class classification problem uses the following classification rule [167], where r is the program output.

$$class = \begin{cases} opel & , \quad r \geq 10 \\ saab & , \quad 0 \leq r < 10 \\ bus & , \quad -10 \leq r < 0 \\ van & , \quad r < -10 \end{cases} \tag{4.7}$$

It is likely that the rule is not optimal. However, it is not the focus of this study to optimally set the rule in order to find an optimal solution of the vehicle problem.

4.7.3.5 other genetic parameters and termination criteria

The ramped half-and-half method is used to create new programs with the maximum depth of four. The crossover rate, the mutation rate, and the reproduction rate are 85%, 10%, and 5% respectively. Also the best of a population is copied once to the next generation. A run is terminated when the number of generations reaches the pre-defined maximum of 200, or the problem has been solved (there is a program with a fitness of zero on the training data set), or the error rate on the validation set starts increasing (for classification problems).

4.7.3.6 experiment configuration

There were six population sizes and four tournament sizes for each of the three problems, giving 72 experiments in total. Each experiment repeated the evolutionary process 100 times randomly and independently, giving a total number of 7200 runs.

4.7.4 Results and analysis

This section presents the experimental results. The analysis considered the ratio of GPPs to the total evaluated programs in a GP run to investigate whether there is only a small fraction of programs relating to the success of finding the

best program during evolution. The analysis also considered the relationships between the GPP ratio and the three factors — tournament size, population size, and problem difficulty — to investigate whether the GPP ratio is influenced by any of these factors. Note that the Bivariate Correlation Analysis [107] is used to provide empirical evidences for supporting or rejecting those relationships. The test calculates a correlation coefficient ($r \in [-1, 1]$) between two variables. $r = -1$ suggests two variables have a perfect negative correlation, $r = 1$ suggests a perfect positive correlation, and $r = 0$ suggests no correlation at all. The confidence level is calculated from the probability value (p). The analysis adopted the commonly used 95% confidence level, meaning that p does not exceed 0.05.

4.7.4.1 the average GPP ratio

Table 4.16 gives detailed results summarising the mean and standard deviation (shown after the $\pm$ sign) of the GPP ratio over 100 runs of each of the 72 experiments. The table is organised into four parts according to the four tournament sizes. Within each part, population sizes are shown as rows and problems are shown as columns. For example, the first cell of the top-left part of the table shows that, for the symbolic regression problem with a tournament size of 20 and a population size of 100, on average, only 15.13% of programs are GPPs amongst the total evaluated programs.

Table 4.16: Average ratio of GPPs to all programs evaluated (%).

Pop Size	Regression	BUPA	Vehicle
	tournament size 20		
100	15.13 ± 7.89	10.70 ± 6.99	12.52 ± 6.95
200	13.62 ± 7.06	10.33 ± 6.03	11.55 ± 6.38
500	9.25 ± 4.71	7.72 ± 4.14	8.91 ± 4.77
1000	8.90 ± 4.18	6.86 ± 3.44	6.68 ± 2.98
2000	6.85 ± 3.55	5.48 ± 3.17	4.92 ± 2.37
5000	**3.89** ± 3.22	3.99 ± 2.14	4.35 ± 1.77

	Regression	BUPA	Vehicle
	tournament size 10		
100	16.95 ± 7.69	13.25 ± 8.04	15.64 ± 7.28
200	15.09 ± 7.02	12.47 ± 6.04	13.09 ± 6.14
500	12.50 ± 5.19	9.78 ± 3.98	11.11 ± 4.28
1000	11.95 ± 4.46	8.14 ± 3.47	10.35 ± 3.67
2000	9.40 ± 4.01	7.99 ± 2.62	9.25 ± 2.47
5000	6.46 ± 3.88	7.42 ± 2.53	8.86 ± 2.11

	tournament size 4		
100	23.30 ± 7.41	18.69 ± 7.60	22.53 ± 7.65
200	23.40 ± 5.68	19.82 ± 6.71	22.41 ± 5.44
500	23.20 ± 4.97	19.11 ± 5.17	21.31 ± 4.50
1000	22.90 ± 4.15	18.81 ± 4.94	21.01 ± 4.56
2000	19.20 ± 6.05	19.56 ± 4.64	21.71 ± 3.53
5000	16.38 ± 6.77	18.38 ± 4.85	21.44 ± 3.42

	tournament size 1		
100	30.78 ± 14.20	30.97 ± 14.81	**33.30** ± 13.09
200	29.41 ± 15.62	29.29 ± 13.17	32.67 ± 12.91
500	26.49 ± 15.55	28.74 ± 14.91	27.86 ± 15.92
1000	23.76 ± 16.64	24.04 ± 15.45	31.56 ± 14.05
2000	25.03 ± 16.36	26.22 ± 14.18	30.95 ± 15.09
5000	21.10 ± 15.94	23.02 ± 15.41	32.51 ± 14.02

Figure 4.6 illustrates several randomly-chosen sample runs from the experiments of the three problems for each of the six population sizes with the arbitrarily-chosen tournament size of 20. There are six charts in the figure corresponding to the six different population sizes respectively. Each chart shows a run for each of the three problems, where the dash line stands for Regression, the thin solid line stands for BUPA, and the thick solid line stands for Vehicle.

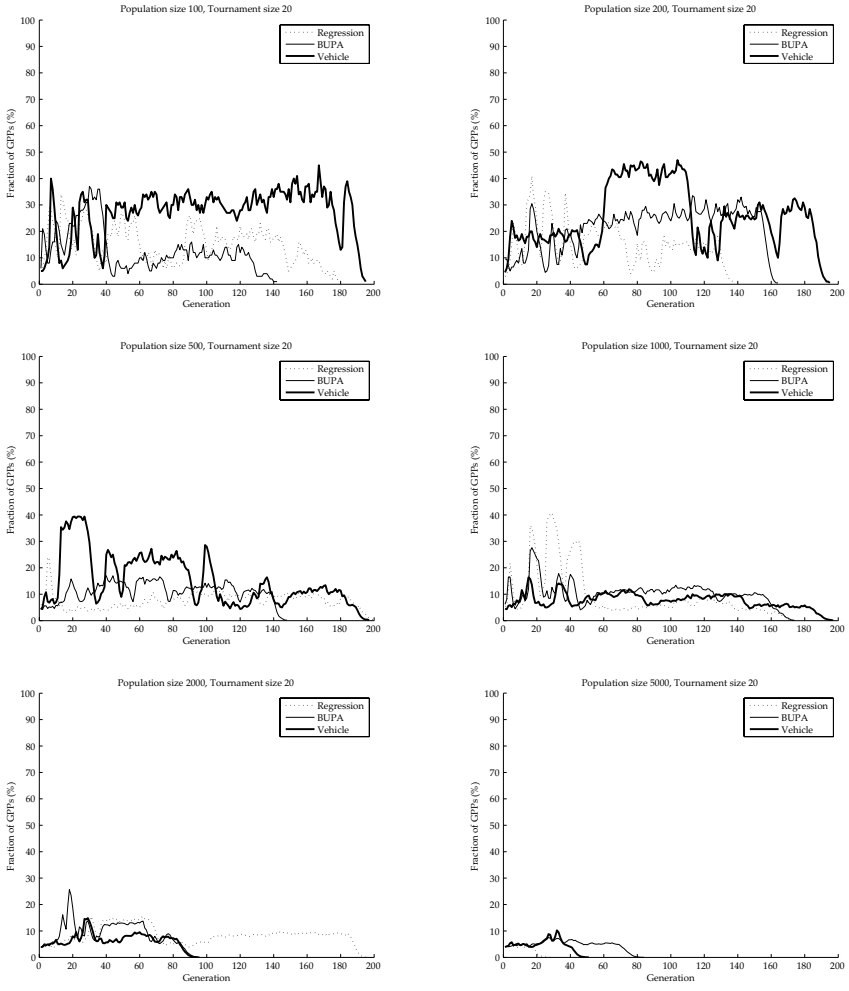


Figure 4.6: Example runs with tournament size 20 for Regression, BUPA, and Vehicle problems using six different population sizes.

From a preliminary consideration of this data, it is clear that many of the evaluated programs do not contribute to the best program in a run. For random selection (tournament size of 1) the GPP ratio can frequently rise to about 46%[5], but

[5]The highest average ratio marked bold in the table is 33.30% and the corresponding standard deviation is 13.09%. Supposing the GPP ratio follows the normal distribution, statistically about 84.1% of runs have the GPP ratio below $33.30\% + 13.09\% \approx 46\%$.

with some parent selection pressure (tournament size of 4), this drops to below 30%, even with small population sizes. With large populations and high parent selection pressure (large tournament sizes), the GPP ratio dropped below 10% for most runs. Therefore, there are many programs whose fitness was evaluated "unnecessarily". Being able to identify these programs before evaluating their fitness could reduce the total fitness evaluation cost very significantly.

It is also clear that the GPP ratio varies with different parameters. To explore the effect of tournament size, population size, and problem difficulty, Figure 4.7 presents a plot of the GPP ratios against population size for each problem category and for each tournament size. Further examination of this data led to the relationships discussed below.

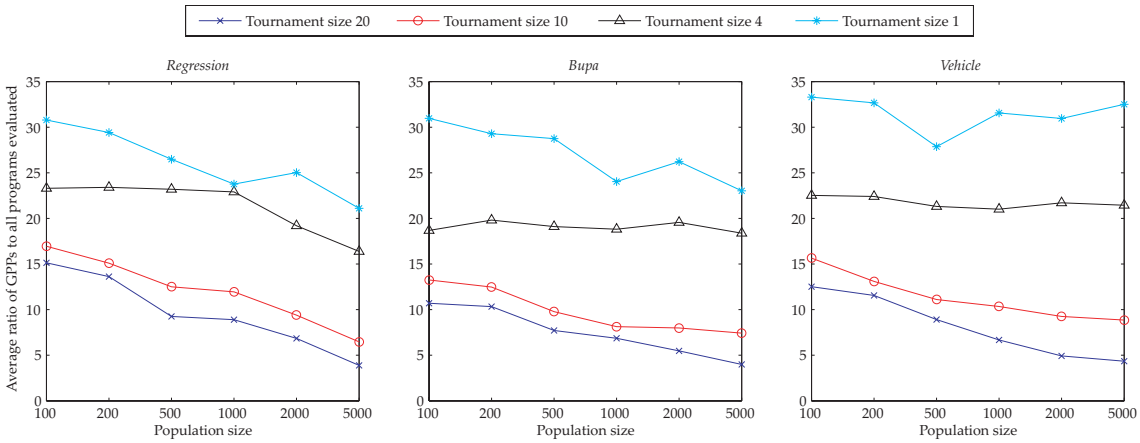


Figure 4.7: Average GPP ratio against population size for each problem and tournament size.

4.7.4.2 GPP ratio and tournament size

The four lines in each graph of Figure 4.7 clearly show that the GPP ratio decreases with increasing tournament size for all problems and all population sizes. Bivariate Correlation Analysis gives a strong negative correlation ($r = -0.852$) between tournament size and the GPP ratio, and the correlation is significant at a 0.01 level. Hence the relationship between the GPP ratio and tournament size is statistically supported.

This is because bigger tournament sizes decrease the chance of any low fitness program winning a tournament and contributing to the next generation. The ancestors of the next generation are likely to be confined to a small set of high

fitness programs, many of which will win multiple tournaments and dominate the new population, and hence constrain the set of GPPs to be small relative to the whole population.

Interestingly, of the four different tournament sizes, the results demonstrate that size 4 has a special characteristic: the GPP ratios across different population sizes in experiments with the tournament size 4 are more stable than those in experiments with other tournament sizes on all three problems. This suggests that a tournament size around 4 may be able to provide a more consistent evolutionary process, regardless of population size, than any of the other tournament sizes. This is probably why tournament size 4 has been very commonly used in many applications.

4.7.4.3 GPP ratio and population size

The relationship between the GPP ratio and population size is not as clear as that between the GPP ratio and tournament size from Figure 4.7. Bivariate correlation gives a correlation coefficient of -0.219 and the significance level is 0.065, indicating a less robust negative correlation between the GPP ratio and population size. However, the negative correlation between the GPP ratio and population size is much stronger for larger tournaments (20 or 10) across all three problems. For the smaller tournaments (4 or 1), the lines are either fluctuating or almost flat. The results suggest that there is a negative correlation between the GPP ratio and population size but the correlation is masked at small tournament sizes.

This might be because larger populations can provide more diverse genetic material, which helps the search find solutions within smaller number of generations, leading to lower ratios of GPPs. However further investigation needs to be carried out.

4.7.4.4 GPP ratio and problem difficulty

As stated in section 4.7.3, Regression, BUPA and Vehicle represent three levels of difficulties — low, medium and high. Figure 4.7 shows no clear relationship between the GPP ratio and problem difficulty. Furthermore, Bivariate Correlation Analysis yields a correlation coefficient of 0.044 (where 0 means no correlation at

all) and insignificant at the 0.717 level (where the significance level should be less than 0.05). Therefore no correlation was suggested between the GPP ratio and problem difficulty in our experiments.

The result is surprising. Assuming a given problem is very easy and a perfect solution can be found in the initial population of size 1000, the GPP ratio will be 0.1%. On the other hand, assuming a given problem is very hard and the search keeps finding better solutions but fails to find the optimal solution when reaching the predefined maximum number of generations, the GPP ratio will be certainly higher than 0.1%. Therefore, the problem difficulty should have impact on the GPP ratio.

Possible explanations for the result are:

- The three problems represent only a small sample of coarse levels of problem difficulties, and may not be sufficient to yield a sound correlation.

- The correlation with problem difficulty may be masked by other factors, including other uninvestigated genetic parameters, effectiveness of fitness functions, and effectiveness of genetic operators.

Therefore further experiments are required to determine whether there is any correlation between problem difficulty.

The analysis of the ratio of the GPPs shows that a small fraction of programs amongst the total evaluated programs in a run are ancestors of the best program. With high selection pressure and large population sizes, the fraction can be smaller than 4% of total programs in some cases. Even with no selection pressure, the fraction can be as low as 33% in average. The results show that it is worth trying to identify and use GPPs to significantly reduce fitness evaluation cost in the parent selection phase.

As GP is a stochastic search algorithm, we do not expect to find clear rules for identifying GPPs. While identifying GPPs prior to the fitness evaluation is difficult, it might be feasible to identify correlates of GPPs and develop mechanisms to use the correlates in the parent selection phase.

4.8 Chapter Summary

This chapter has investigated ways to improve the efficiency of tournament selection for parents. It introduced Ejit, which was based on the not-sampled characteristic of the standard tournament selection, followed by an analysis of its limitations. It presented a simple but novel population clustering algorithm — the heuristic fitness-case-equivalence population clustering algorithm — for GP in order to take the advantage of the clustering tournament selection. The effectiveness as well as the limitations of the algorithm has been illustrated through the experimental analyses. The limitations of Ejit and the population clustering algorithm motivated another thought about *minimising* the fitness evaluations in the tournament selection for parents. Therefore, this chapter also conducted an initial analysis on the feasibility of using GPPs to minimise the fitness evaluation cost in the tournament selection for parents. The detailed outcomes are as follows.

- Ejit avoids the evaluations of not-sampled individuals to reduce the fitness evaluation cost for standard tournament selection. Unlike EMS-EA and BC-EA, it does not require any extra memory, or any pre- or post-processes. In Ejit, the decision on which program should be evaluated arises naturally from using a passive evaluation order. Ejit is expected to provide savings about 37%, 14%, 5%, 1.8% and 0.7% for tournament size 1 to 5 respectively. The expectation was verified by the experiment results. Although Ejit works only with standard tournament selection and provides only limited saving for larger tournament sizes, its features make it significantly attractive for hard problems that require very low parent selection pressure.

- The potential of usefulness of the heuristic fitness-case-equivalence population clustering algorithm has been illustrated via the SymReg problem. However, this did not work well for BinCla because it is not necessary to precisely cluster a population into fitness-case-equivalences. Developing a more robust (or fuzzier) fitness-case-equivalence population clustering algorithm together with the clustering tournament selection is a promising direction to improve the efficiency of the tournament selection for parents and is worth further investigation.

- The heuristic fitness-case-equivalence population clustering algorithm seems to be inappropriate for EvePar and other Boolean problems, where there are only two program outputs (*true* or *false*) that are not sufficient for clustering a population properly. To overcome the problem, the genotype population clustering algorithm has been proposed and tested. The experimental results show that with the genotype population clustering algorithm, not only can the clustering tournament selection be used to significantly improve the search performance under high parent selection pressure, but also the saving on the number of fully-evaluated programs can be considerably greater than Ejit for large tournament sizes. The literature reveals there are many other genotype related measurements [25, 44, 60, 120, 132] that can be used to cluster a population, and that are worthy of further investigation.

- A framework was developed to gather information on GPPs. A series of experiments was conducted to test the hypothesis that only a small fraction of programs amongst the total evaluated programs in a run are ancestors of the best program. The analysis of the ratio of the GPPs shows that with high parent selection pressure and large population sizes the hypothesis clearly holds. The number of GPPs is smaller than 4% of the total evaluated programs in some cases. The analysis of the relationships between the GPP ratio and three factors —tournament size, population size, and problem difficulty — show that the GPP ratio is strongly influenced negatively by tournament size, and also by population size but less strongly, but is not influenced by problem difficulty.

It was surprising that there was no evidence for a correlation between the GPP ratio and problem difficulty in the experiments. Future work could choose more problems representing finer difficulty levels in order to further investigate the relationship between the GPP ratio and problem difficulty.

While it seems that so far it would be challenging to develop a new approach that can identify the GPPs directly in advance, thus avoiding the fitness evaluations on non-GPPs, it would be worth making further effort to explore ways to identify the correlates of GPPs, as once it can be done, the fitness evaluation

cost can be reduced as much as possible without affecting the effectiveness of the current GP system.

Part II

Analysing Impact of Offspring Selection

Chapter 5

Applying Offspring Selection Pressure

The first part of this thesis focused on tournament selection to investigate strategies to control parent selection pressure and ways to reduce the computational cost in the parent selection phase. The second part of this thesis focuses on offspring selection to investigate impacts of offspring selection on the overall GP search performance and heuristics for constraining offspring search space based on program structure.

This chapter firstly makes the assumption that there exist constructive operators, which can result in good offspring directly without extra computational cost, and therefore introduce offspring selection pressure and reduce the stochastic nature of GP. It then presents a set of experiments to investigate the impact of such constructive operators on the overall GP performance and to investigate how to properly configure the selection pressure between the parent and offspring selection.

Since only simulations of constructive operators using local search are used rather than real constructive operators themselves, this chapter then considers the extra computational cost required in the simulated constructive operators and presents another set of experiments to investigate whether the extra cost is worthwhile in the context of a resource-limited GP search, and how intensive the local search should be.

5.1 Introduction

In the standard GP algorithm, there is no offspring selection; selection pressure is applied only in the selection of parents and the offspring produced is put into the next generation without selection. With the standard breeding process of the GP algorithm, exploring new states in the "neighbourhood"[1] search space of current states can be viewed as a set of random walks.

However, the number of possible offspring in the immediate neighbourhood of any chosen parents is large, and a large fraction of these offspring will not constitute improvement over the parents [133, 134]. Therefore, even an increased number of usages of good parents under high parent selection pressure could still be insufficient to provide a good chance of finding good offspring.

One approach to overcoming this problem is to increase the chance of generating improved offspring from parents by using customised *constructive* genetic operators that *avoid generating worse offspring altogether*. However, designing such operators can be difficult and is likely to be domain-dependent.

An alternative, simpler, and domain-independent approach is to allow the selected parents to produce more offspring via crossover or mutation and to integrate variants of local search techniques into a many-offspring breeding process to search for good offspring [64, 97, 114, 115, 178, 182], replacing the standard breeding process by the many-offspring breeding process. The crossover and mutation operators in the many-offspring breeding process are not strictly constructive operators. At most they can be viewed as simulated constructive operators that produce the same outcomes as constructive operators but will have to generate a large number of poor offspring in the search for good offspring in successor states.

However, the use of simulated constructive operators in the breeding process alters the standard GP search algorithm by increasing the selection pressure towards good offspring and *further* reducing the stochastic nature of the GP search. Although there are some promising results of the use of simulated constructive operators from the literature, it is still not exactly clear how increasing selection

[1] The distance between these states may be very large.

pressure in the breeding process affects the overall GP search performance. This is because, in general, increasing selection pressure tends to confine the search process, speed up the loss of population diversity, and lead the search to premature convergence or other undesirable situations. It is necessary to explore the actual effect of the offspring selection in combination with parent selection.

The key element in simulated constructive operators is local search. The more intensive the local search, the greater the selection pressure in the choice of offspring, and the smaller the stochastic element in the breeding process: a very intensive search can generate the best possible offspring of given parents; a less intensive search that only considers part of the neighbourhood will have a greater stochastic element and may generate offspring that are good but not the best possible. From the literature, most approaches use partial local searches that consider only subsets of the immediate neighbourhood of the chosen parents. Therefore, it is important to understand what would happen if extending this to a complete local search that considers all possible offspring.

As discussed in Chapter 2 (on page 39), crossover has remained the dominant genetic operator in deriving optimal solutions. As a result, the second part of this thesis will focus on crossover when conducting investigations and analyses related to offspring selection.

5.2 Chapter Goals

This chapter aims to determine whether it is worth trying to design constructive operators, and how offspring selection will affect the GP search performance.

There is no question that an intensive local search for good offspring of given parents is expensive and will take resources away from exploring more possible parents and more generations of programs. Therefore, in addition, this chapter addresses the effectiveness of a many-offspring breeding process as a technique in its own right as part of the GP process. It explores whether the cost of such a local search can be worthwhile in the context of a resource-limited GP process, and investigates the appropriate intensity of a local search in the breeding process to maximise the performance of a GP system.

5.3 Simulations of Constructive Crossover Operators

Investigating the research questions requires building at least two simulations of a constructive crossover operator which embody different intensities of local search. One simulation mimics an "*ideal*" constructive crossover operator that produces the best offspring for a given pair of parents. The simulation considers all possible ways of recombining two chosen parents to produce *all* possible offspring, then evaluates them, and keeps two offspring with the best fitness values but throws away the others. We refer to it as *Full Xover*. It is similar to the brood recombination crossover [178] but does not have a brood size to restrict the search of good offspring. A second simulation mimics a *partial* constructive crossover operator that produces good offspring but not necessarily *the best* for a given pair of parents. It chooses a crossover point randomly in one parent P_1 but considers all nodes in the other parent P_2 to produce possible offspring, then evaluates them, and keeps the top two. It is similar to the *context-aware crossover operator* [115] but has no constraint on depth while choosing possible crossover points in P_2. We refer to it as *Partial Xover*. Both simulations focus on optimising the immediate offspring's fitness — problem-solving quality — but Partial Xover contains some stochastic elements while Full Xover completely eliminates them.

5.4 Experiment Design

To investigate the effect of including offspring selection pressure in GP search, our experiments consider six different combinations of selection pressure illustrated in Figure 5.1. The selection of parent programs has two options, either without selection pressure by using a random parent selection process, or with selection pressure, as in the standard GP algorithm. The selection of offspring in the breeding process can have three levels of selection pressure: no selection pressure in the standard breeding process, or weak selection pressure using Partial Xover, or strong selection pressure using Full Xover.

The experiments explore the consequences of the six different combinations on three different domains — EvePar, SymReg, and BinCla (see page 76 for details

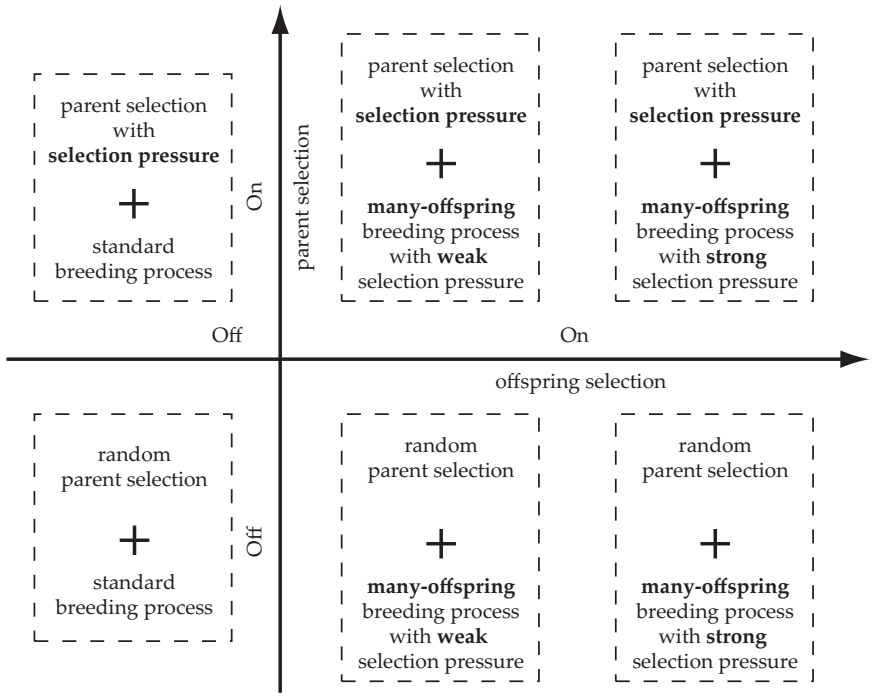


Figure 5.1: Six GP systems according to configurations of selection pressure on parent selection and offspring selection.

about the problems, the fitness functions, and the terminal and function sets.).

5.4.1 Genetic parameters

The genetic parameters are the same for all three problems. The ramped half-and-half method is used to create new programs with the maximum depth of four. The population size is 100. The crossover rate and the reproduction rate are 95% and 5%. For ease of analysis, the mutation operator is not used. The maximum size of a program is 50 nodes based on some initial experimental results.

Standard tournament selection is used to select parents. Selection pressure on the parent selection is switched on or off by setting the tournament size to 4 or 1 respectively. Tournament size of 4 is chosen based on empirical search. We expect these settings to provide a neutral environment when conducting performance comparisons. We also apply a selection policy for selecting parents and selecting offspring in Partial Xover and Full Xover. The selection policy consists of three measures: fitness value, number of nodes, and depth of tree. The three measures are applied sequentially in order to select a program with a shallower tree depth and a smaller number of nodes if its fitness value is the same as its competitors.

5.4.2 Experiment configuration

We performed two sets of experiments with different termination criteria. The first set of experiments (Exp1) treats the simulated constructive crossover operators as if they were real constructive operators, so that the cost of performing the local search for the best offspring is ignored. The runs are terminated when the number of generations reaches the pre-defined maximum of 51 (including the initial generation), or the problem has been solved.

Computational resources include memory and CPU time. The simulated constructive crossover operators do not require extra memory but do require a large amount of CPU time to generate and test offspring. As a result, the second set of experiments (Exp2), takes into account the cost of the local search, and terminates the runs when the total CPU time (seconds) exceeds a pre-defined limit, or the problem has been solved. The time limits were determined from analysis of the results in Exp1.

We ran experiments comparing GP systems with and without parent selection pressure using the standard crossover operator, the simulated partial crossover operator and the simulated ideal crossover operator respectively for each of the three problems. Each experiment repeated the whole evolutionary process 100 times independently.

In Exp1, an additional termination criterion is applied to BinCla as an overfitting prevention strategy. We split the original BinCla data randomly and equally into a training data set, a validation data set and a test data set. Selection of the best program in a population is based on its fitness on the training data set as the fittest program in the corresponding generation. The fitness values of the best program on the training data set and the validation data set within a moving window of size 15 are monitored to detect overfitting. The window size of 15 is chosen based on empirical search. A run terminates when the training fitness of the latest generation in the window has not been improved over the window. The run also terminates when the validation fitness of the latest generation in the window is not better than that of the earliest generation in the window, indicating an occurrence of overfitting. For a run which stops according to the strategy, we examine the window and select a generation where the fitness on the validation

set is the best in the window. The corresponding test fitness value is used as the performance measure of the run.

For Exp2, the search is expected to continue until it exceeds the given CPU time. The earlier stopping strategy used in Exp1 for preventing overfitting would be inappropriate in Exp2. As cross validation is a common technique to reduce the dangers of overfitting [160], 10-fold cross validation is used for BinCla with attempts to ensure class labels are evenly distributed. The performance measure of a run is the average of the best test fitness value over 10 folds.

5.5 Results and Discussions: Exp1

5.5.1 Effectiveness

Table 5.1: Performance of systems using 3 different crossover modes with/without selection pressure in Exp1.

GP Systems	Parent Selection	Xover Mode	EvePar Failure	SymReg RMS Error	BinCla Test Error Rate (%)
Sys1	On	standard	100%	62.5 ± 3.5	16.4 ± 7.3
Sys2		partial	88%	58.5 ± 3.3	10.9 ± 4.7
Sys3		full	77%	58.8 ± 5.7	9.7 ± 4.3
Sys4	Off	standard	100%	65.4 ± 1.0	16.2 ± 6.3
Sys5		partial	91%	55.4 ± 2.9	8.5 ± 2.6
Sys6		full	0%	37.2 ± 5.7	6.7 ± 2.4

Table 5.1 shows the performance measures of the first set of experiments. When parent selection pressure is switched on, GP systems using Partial and Full Xover (Sys2 and Sys3) outperform the GP system using the standard crossover operator (Sys1). The performances in Sys2 and Sys3 are noticeably different in EvePar but quite similar in SymReg and BinCla.

From the results, it seems that reducing stochastic elements in the breeding process does not have negative effects and the constructive crossover operator is effective. This observation matches those made by proponents of a many-offspring breeding process.

However, interesting results are obtained after comparing the performances of the GP systems with and without parent selection pressure:

- Sys1 and Sys4, using the standard crossover operator, have the similar worst performance on all three problems.
- For EvePar, Sys2 and Sys5 have similar failure rates, but Sys2 is slightly lower than Sys5. Sys6 using Full Xover without parent selection pressure is much better than Sys3. All 100 runs successfully found the optimal solution in Sys6.
- For SymReg and BinCla, Sys5 and Sys6 are better than Sys2 and Sys3 respectively.
- Overall, the best performance is obtained by Sys6 using the simulated ideal constructive crossover operator *without* parent selection pressure.

These results suggest that premature convergence may occur more often in Sys3 than that in Sys6. To confirm this, we examined the index of the generation where the best-of-run appeared for the first time for Sys3 and Sys6. The results are illustrated in Table 5.2. We realised that, for instance in EvePar, the index in Sys3 ($\mu = 14$, $\sigma = 7$) is much earlier than that in Sys6 ($\mu = 21$, $\sigma = 6$), indicating that the GP system with parent selection pressure causes the search to end up with premature convergence more often if stochastic elements are completely removed in the breeding process. Similar phenomena occurred in SymReg and BinCla as well.

Table 5.2: The average index of generation where the best-of-run appeared first time in Sys3 and Sys6 in Exp1.

GP Systems	EvePar	SymReg	BinCla
Sys3	14 ± 7	12 ± 6	6 ± 4
Sys6	21 ± 6	47 ± 4	16 ± 6

The maximum number of generations stopping criterion in Exp1 can be seen as setting a limited number of movements in a search process. In this situation, it is better to carefully make a wise movement for each step. The problem-solving quality can be significantly improved generation by generation by always moving to the fittest status, indicating that the hill-climbing metaphor can be applied in the GP search algorithm to improve its performance. If we could have an ideal

constructive crossover operator (not a simulated one so that there is no expensive cost of generating and testing poor offspring), we should certainly use it to replace the standard blind random crossover operator, and most importantly, we should certainly remove selection pressure from the parent selection. On the other hand, if we do not have the ideal constructive crossover operator and have to use the expensive generate-and-test process to find good offspring, but the effectiveness is critical and the efficiency can be less important, we should also replace the standard breeding process and turn off the parent selection pressure.

5.5.2 Efficiency

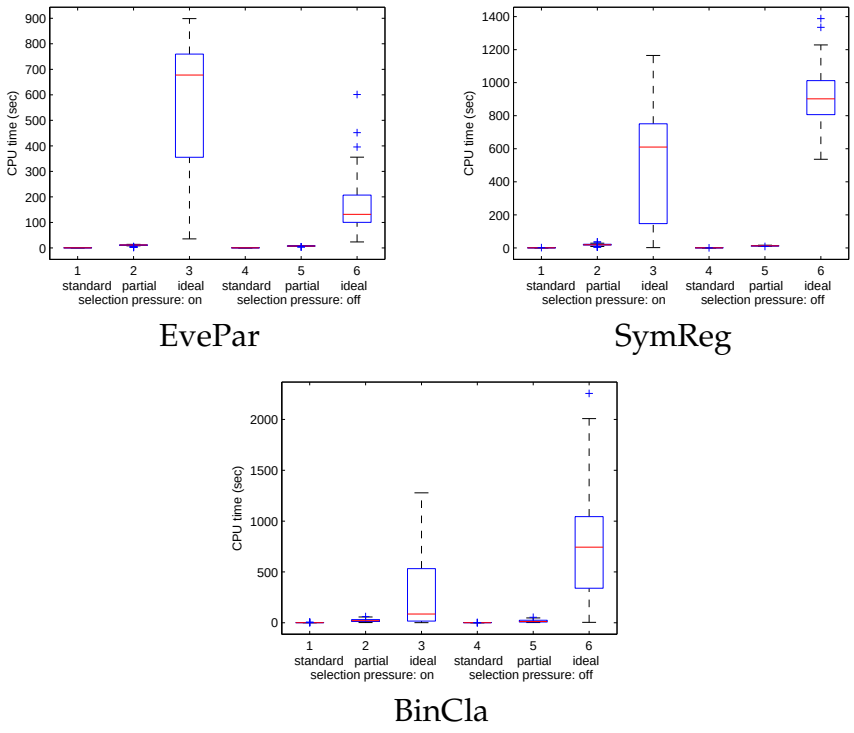


Figure 5.2: Boxplot of CPU time consumed in systems in Exp1.

Figure 5.2 shows the boxplots of CPU time consumed by 100 runs in each of the GP systems for the three problems. It is clear that the systems using Full Xover required much more CPU time than the others as they needed to evaluate a huge number of offspring.

In EvePar, 77 runs in Sys3 using the Full Xover with parent selection pressure switched on must keep searching until the 51st generation, while all runs in Sys6

with parent selection pressure switched off stop early as optimal solutions are found. Therefore it is not surprising that runs in Sys3 consumed much more time than those in Sys6.

In SymReg, as all runs stop at the 51st generation, we expected a similar amount of CPU time consumed for runs in systems using the same crossover mode. However, runs in Sys3 surprisingly consumed less time than those in Sys6. We then calculated the number of programs evaluated during evolution in Sys3 and Sys6 and found that the number of evaluated programs in Sys3 ($\mu = 1.00 \times 10^6$, $\sigma = 0.58 \times 10^6$) is much smaller than that in Sys6 ($\mu = 2.14 \times 10^6$, $\sigma = 0.27 \times 10^6$). This is because parent selection pressure forces the search to focus on fit and smaller size programs. The positive effect is that the system controls the code bloat by effectively filtering out programs containing introns. Therefore, in Sys3, with smaller program size, the number of offspring evaluated is also smaller than that in Sys6 because the local search space is much smaller with smaller programs, resulting in less CPU time used. However, a side effect is that the system also abandons currently unfit subprograms that possibly will be useful later and reduces the population diversity.

In BinCla, due to the use of the overfitting preventing strategy, runs often stop before the 51st generation. The average total number of generations used over 100 runs in Sys3 ($\mu = 19$, $\sigma = 3$) is much smaller than that in Sys6 ($\mu = 29$, $\sigma = 6$). When considering the problem-solving quality of Sys3, the smaller number of generations used is possibly due to the early occurrences of local optima. Therefore it is not surprising that runs in Sys3 consumed less CPU time than those in Sys6.

5.6 Exp2

The comparisons and suggestions in terms of the effectiveness made above are based on the assumption that we have a constructive crossover operator which can *directly* produce a better or the best offspring without extra computational cost. Thus the actual computational resources required by the simulated constructive operators were irrelevant. The second set of experiments are intended

to investigate the value of the many-offspring breeding process in the absence of a real constructive crossover operator, and have to take this cost into account.

5.6.1 Determining time limits

Sys6 took the largest amount of CPU time in Exp1. To determine appropriate time limits for the runs in the second set of experiments, we identified the maximum CPU time taken by Sys6 for each of the problems in the first set of experiments, ignoring the outlier runs. On the basis, the CPU time limits in Exp2 are 400, 1200, and 2000 seconds for each run in EvePar, SymReg, and BinCla, respectively. For BinCla, as 10-fold cross validation is used, each fold is assigned an equal amount of CPU time (200 seconds). As a 16.40% classification error rate is the worst performance on average in Exp1, the value is used as the threshold in the pruning algorithm in Exp2.

5.6.2 Results

Table 5.3 shows the performance measures of the second set of experiments. For EvePar, runs may terminate before completely consuming the given upper bound CPU time as a result of finding optimal solutions. Therefore, an additional measure, *actual time*, is used for EvePar to measure the actual CPU time in seconds consumed in total. For SymReg and BinCla, there is no optimal solution found and all runs terminate when exceeding the allowed CPU time, so the actual time measure is omitted.

Table 5.3: Performance of systems using 3 different crossover modes with/without parent selection pressure in Exp2.

GP	Parent	Xover	EvePar		SymReg	BinCla
Systems	Selection	Mode	Failure	Actual Time	RMS Error	Test Error
Sys1	On	standard	66%	299 ± 154	53.1 ± 7.6	8.5 ± 1.1
Sys2		partial	68%	288 ± 167	58.6 ± 3.6	7.1 ± 1.0
Sys3		full	78%	338 ± 122	58.0 ± 5.8	7.2 ± 1.0
Sys4	Off	standard	90%	370 ± 94	52.3 ± 5.2	6.9 ± 0.8
Sys5		partial	0%	21 ± 15	38.9 ± 5.0	4.1 ± 0.6
Sys6		full	4%	140 ± 86	37.3 ± 5.6	4.2 ± 0.6

5.6.3 Parent selection pressure: on

When selection pressure is applied to the parent selection, surprisingly, the standard crossover operator is better than the simulated constructive crossover operators in EvePar and SymReg, and has only a slightly lower performance in BinCla. This observation suggests that if sufficient search time is given, when selection pressure is applied to the parent selection for crossover, there is no advantage in using a many-offspring crossover to replace the standard blind random crossover operator. In other words, the standard crossover operator works *quite well* in comparison to a simulated constructive crossover operator using local search.

This observation is different from the observation in Exp1 in the case of a real constructive crossover operator. A possible explanation is that the selection pressure on the parent selection removes some stochastic elements and then forces the search to focus on a smaller region. When stochastic elements, which are necessary in order to find global optima, are further removed in the many-offspring breeding process, the search will eventually end up at local optima. In contrast, the standard breeding process keeps some stochastic elements which help the search escape local optima in a limited amount of search time.

5.6.4 Parent selection pressure: off

When parent selection pressure is switched off, Sys5 and Sys6 both produce significant performance improvements. In EvePar, Sys5 outperforms the other two, not only by the zero percent failure rate, but also by the much shorter CPU time consumed (about 21 seconds on average). Note that the failure rates in Sys6 are different between Exp1 and Exp2. This is possibly because some outliers in Exp1 require more CPU time to complete their search but the CPU time set in Exp2 terminates those outliers before they find optimal solutions. In the other two problems, Partial Xover and Full Xover have similar performance.

The results suggest that when stochastic elements are fully preserved in selecting parents, it is necessary to conduct an intensive search in the successor states of chosen parents in order to remove the stochastic elements and make good movements so that the search will act differently from a completely random search.

The results for EvePar suggest using Partial Xover instead of Full Xover. One possible explanation is that for the search to make as many movements as possible within the given time frame, it would be better merely to look at a subset of all possible movements so that a larger number of less perfect movements can reach the goal faster than a smaller number of perfect movements. But the subset has to be sufficient large to cover most important movements. The results suggest Partial Xover may lead the search into an optimal subset of successor states. From the exploration vs. exploitation [42] point of view, Full Xover spends most of its time exploiting the known genetic material and less time exploring other potential useful search space. Therefore, if a tight time-frame is given, it may often fail. To confirm this, we chose a new boundary of 55 seconds, which is only about two standard deviations away from the mean in Sys5, and re-examined the failure rates in Sys5 and Sys6. With the new time boundary, Full Xover rose from 4% to 89% runs, exceeding the time limit without finding optimal solutions, while Partial Xover had only about 4% runs that failed. The standard crossover is exactly opposite to Full Xover. It puts too much effort into exploring the search space and little effort into exploiting the known genetic material; thus it also fails.

5.6.5 Overall

From comparing the performances of GP systems with and without selection pressure applied to parent selection, we conclude that for EvePar, the performance of Sys4 is the worst, but for SymReg and BinCla, it is slightly better than or comparable with Sys1, Sys2, and Sys3 where parent selection pressure was turned on. This observation suggests that, if enough search time is given, a random (beam) search on parent selection can have a similar problem-solving quality to GP systems with selection pressure applied to the parent selection.

We also conclude that Sys5 and Sys6 are outstanding in all six systems. The results strongly suggest that if we intend to use a many-offspring crossover instead of the standard one, we should remove the selection pressure from the parent selection to avoid premature convergence in order to gain further performance improvement.

Table 5.4: Performance of systems with population size of **1000** using 3 different crossover modes.

GP Systems	Parent Selection	Xover Mode	EvePar		SymReg	BinCla
			Failure	Actual Time	RMS Error	Test Error
Sys1	On	standard	20%	125 ± 155	45.2 ± 7.0	5.3 ± 0.7
Sys2		partial	10%	70 ± 113	45.5 ± 4.7	4.0 ± 0.7
Sys3		full	46%	381 ± 45	40.8 ± 4.4	4.0 ± 0.5

5.6.6 Further discussion

It might be argued that it is not necessary to turn off the parent selection pressure when using a many-offspring breeding process, since the population diversity could be easily maintained by increasing the population size while keeping selection pressure on the parent selection. To verify whether this argument is true, we conducted another set of experiments for Sys1, Sys2, and Sys3. In the third set of experiments, the population size was increased from 100 to 1000; other parameters and stopping criteria, including the CPU time settings, were the same as those in Exp2. Table 5.4 lists the results.

By comparing the performances of corresponding GP systems in Tables 5.3 and 5.4, it is clear that the problem-solving qualities are improved with a larger population size. However, by also considering the performances of Sys5 and Sys6 in Table 5.3, it is clear that, within the same CPU time limit, the improvements obtained by increasing the population size by a factor of 10 are not as significant as, or are very similar to, those obtained by just switching off parent selection pressure.

Note that population diversity could be maintained by mutation. Using a high mutation rate such as 20% might provide different results. It would be necessary to consider mutation operators in further work.

5.7 Chapter Summary

Stochastic elements exist in both the parent selection process and the breeding process. Some stochastic elements need to be removed in order to distinguish the genetic search algorithm from a random search algorithm. On the other hand some stochastic elements must be retained in order to prevent the genetic search

from being confined in local optima or converging prematurely.

Selection pressure on the parent selection removes some stochastic elements. After local search techniques are integrated into the breeding process, stochastic elements are further eliminated. The change was suggested as effective in the literature [64, 97, 114, 115, 178, 182]. However, this chapter obtained different results after investigating six GP systems involving two simulations of constructive crossover operators with/without parent selection pressure. The detailed outcomes are follows:

- Increasing the selection pressure towards good offspring is better than randomly generating offspring. It is worth trying to develop constructive operators. However, the parent selection pressure should be reduced in order to take the advantage of the offspring selection and to significantly improve the GP search. Otherwise, premature convergence may often occur. Our experimental results show that it is better to apply selection pressure to offspring selection than to parent selection.

- In the context of a resource-limited GP process, if stochastic elements are minimised or optimised in the parent selection process, for instance tuning the tournament size, it is better to keep some stochastic elements in the breeding process, for instance using the standard blind random crossover operator. On the other hand, if stochastic elements are minimised or optimised in the breeding process, for instance performing enough intensive searching in successor states of chosen parents, then it is better to keep some stochastic elements in the parent selection process, for instance selecting parents randomly for crossover. Stochastic elements cannot be removed in both parent selection process and breeding process.

- The effectiveness of a many-offspring breeding process as a technique in its own right as part of the GP process has been demonstrated in the context of a resource-limited GP process when the parent selection pressure is turned off. Further, using a larger population size with parent selection pressure is not better than using smaller population size without parent selection when using either Partial or Full Xover.

- When no selection pressure is applied to parent selection, a proper level of local search intensity can be expressed by Partial Xover instead of the exhaustive Full Xover in our experiments. Further investigations are required in order to make a more general conclusion.

Overall, if offspring selection is applied, no matter whether real constructive crossover operators or their simulations are used, the parent selection pressure needs to be reduced in order to obtain significant search performance improvement.

This chapter investigated only the crossover operator in the context of many-offspring breeding. In the future, similar studies on mutation, as well as the use of a combination of crossover and mutation, should be conducted in order to provide a complete picture.

From the literature, the commonest method for searching good offspring is generate-and-test, which is very expensive. In order to make the use of offspring selection more beneficial, it would be really useful to have heuristics that can be applied to reduce the offspring search space. The next chapter will address this topic.

Chapter 6

Constraining Offspring Search Space

This chapter continues to focus on crossover to investigate heuristics for improving the efficiency of searching good offspring. It firstly presents several simulations of the optimal crossover operator. It then analyses these simulations with a focus on the depth of crossover point and the substituted subtree size to determine whether any patterns exist. Finally it presents some heuristics that could be used to reduce the offspring search space.

6.1 Introduction

The previous chapter showed that performing offspring selection with a bias towards better fitness offspring after randomly selecting parents was preferable to other combinations of parent and offspring selection strategies. However, the offspring search space for a given pair of parents could be enormously large. In the case of crossover, the size is limited — its maximum is just the product of the numbers of nodes in each parent — but finding good crossover points is still very expensive. In the literature, a common approach to finding good crossover points is the generate-and-test method. The method is really time consuming and takes lots of resources on a large fraction of poor offspring, although some researchers proposed to use a fraction of training data for offspring fitness evaluation [4, 178, 56]. Therefore, it is necessary to develop some good heuristics that can be used to locate good crossover points, or at least eliminate bad crossover points, with minimal cost in the large offspring search space of given parents.

One possible kind of heuristic is to constrain the syntax and types of swapped subtrees [33, 40, 59, 66, 118] so that offspring search space can be reduced. Another approach is to apply constraints based on program structure, as in the large number of studies related to depth and size controlling strategies in crossover since the 1990s (see page 35). From the literature, while existing depth or size control strategies can sometimes improve the GP performance, the number of such different strategies is large and identifying a good strategy for efficiently narrowing the set of candidate crossover points remains a challenging problem.

This chapter investigates heuristics for locating good crossover points from a different angle. A desired characteristic of a really effective and efficient GP search algorithm is that its GP search path can always reach the optimal solution for a given problem and the path is the shortest. Therefore, in order to effectively reduce the offspring search space, we should analyse the characteristics of the *optimal* crossover operator that always chooses the crossover leading most directly to the optimal solution. We expect the behaviour patterns of the optimal crossover operator to provide the best insight into good heuristics on program structure for selecting crossover points in ordinary crossover operators.

Note that the optimal crossover operator should differ from the ideal constructive crossover described in the previous chapter. An optimal crossover operator produces good offspring in a global-wise manner, while an ideal constructive crossover operator produces good offspring in a stepwise manner, that is, it concentrates on producing only good immediate offspring, which may not necessarily lead to optimal solutions most directly.

6.2 Chapter Goals

This chapter focuses on two aspects of program structure, including the depth of crossover point (DCP) and the substituted subtree size (SSS), and aims to investigate general heuristics for efficiently finding good offspring by constraining crossover point selection structurally through the analysis of the characteristics of optimal crossover operators.

The chapter firstly analyses the DCP in good crossover events and particularly

addresses the following research questions:

- How should the DCP be adjusted in order to significantly improve GP performance?
- Is there a certain range of depths where crossover points fall such that good offspring can be consistently produced and consequently the overall GP search performance can be optimised?

The chapter then examines the distributions of SSS in good crossover events to investigate whether there are any patterns of the substituted subtree sizes. In particular, it investigates whether good crossover events consistently involve subtrees within certain size ranges. This is important because if such ranges of subtree sizes exist, higher priorities can be assigned to subtrees whose sizes are within the ranges and lower priorities can be assigned to subtrees with other sizes. When searching in the offspring space for given parents, the search path can follow the priorities to find good offspring instead of searching randomly or exhaustively.

6.3 Our Approach

Ideally, to answer the research questions, we should analyse the behaviour of the optimal crossover operator. However, the optimal crossover operator cannot be implemented. The conceptually simplest simulation of the optimal crossover operator is to perform an exhaustive search of all possible crossovers in an evolution from an initial population and identify all the crossovers on the paths leading to the optimal solution. This set of crossovers would contain all the steps that the optimal crossover operator would follow, and we could then analyse DCP and SSS on these steps.

However, an exhaustive search of all sets of crossovers is completely infeasible, even for the most trivial of problems. We therefore investigate a sequence of approximations to the simulation of the optimal crossover operator. Each approximation considers a large set of possible crossovers of each pair of parents, and greedily chooses the best offspring. At the end of a run, we identify the parents

and crossovers that contributed to the best solution found, on the assumption that these are close to the crossovers that would be performed by the optimal crossover operator. Our analysis will search for patterns in these crossovers.

Our simulated optimal crossover operators are inspired by brood recombination [178]. The operators produce offspring but consider a large number of the possible crossover positions, including the root node, in two chosen parents to produce children. For each parent, the operators then evaluate the children that were obtained by substituting a subtree from the other parent into this parent and retain the best-performing one, as measured by fitness. Therefore, the two retained children would not necessarily be a pair of children produced by swapping subtrees. This is also the reason that this chapter uses a more general term *substituted subtrees* instead of the commonly used *swapped subtrees*. Since the simulations consider only the fitness of the immediate offspring rather than the fitness of the best descendant, they can be only approximations to the optimal crossover operator.

The degree of approximation varies with the fraction of possible crossover positions that are considered, which we refer to as the *search intensity* of the operator. Obviously, Standard Xover has the lowest possible search intensity. The simulated optimal crossover operator that considers all possible crossover positions ($n \times m$ if the parents have n and m nodes respectively) has the highest search intensity (Full Xover). The other approximations choose only some of the possible crossover positions in the two parents (P_1 and P_2):

- *Partial$^+$ Xover*, which considers every node in P_1 combined with $\frac{\sqrt{m}}{2}$ random nodes from P_2, and every node in P_2 combined with $\frac{\sqrt{n}}{2}$ random nodes from P_1 to generate $\frac{n\sqrt{m}+m\sqrt{n}}{2}$ offspring.
- *Partial Xover*, which considers one randomly-chosen node in P_1 combined with every node from P_2 to generate m offspring.
- *Partial$^-$ Xover*, which considers one randomly-chosen node in P_1 combined with $\frac{\sqrt{m}}{2}$ random nodes from P_2 and one randomly-chosen node in P_2 combined with $\frac{\sqrt{n}}{2}$ random nodes from P_1 to generate $\frac{\sqrt{m}+\sqrt{n}}{2}$ offspring.

If the two parents have the same size of n nodes, then the search intensities of

Full Xover, Partial$^+$ Xover, Partial Xover, and Partial$^-$ Xover are 1, $\frac{1}{\sqrt{n}}$, $\frac{1}{n}$, and $\frac{1}{n\sqrt{n}}$ respectively.

In this study, we split crossover events into four categories since the root of a program tree is a potential crossover point. The first category is *subtree-subtree*, which is the normal strict subtree to strict subtree replacement in crossover in tree-based GP. The second category is *root-root*, where the offspring is produced by using P_2 to replace P_1 or vice versa. It means that the crossover is effectively the same as the copy operator. We term this type of crossover operation as *copy-like* crossover operation. The third category is *root-out*, where the offspring is produced by using a strict subtree from P_2 to replace the entire P_1 or using a strict subtree from P_1 to replace the entire P_2. This is effectively the same as *promotion mutation* in [163]. The fourth category is *root-in*, where the offspring is produced by using the entire P_2 to replace a strict subtree of P_1 or using the entire P_1 to replace a strict subtree of P_2. We term the three types of crossover events involving parent program roots as *root crossover events* although they would not be categorised as crossover according to the usual definition of crossover.

6.4 Experiment Design

Our experiments used the three problems, namely EvePar, SymReg, and BinCla, which have been used in the previous chapters of this thesis. More details about these three problems, the data sets, the fitness functions, the terminal sets, and the function sets can be found on page 76.

The genetic parameters were the same for all three problems. The ramped half-and-half method was used to create new programs with the maximum depth of four. The crossover rate and the copy rate were 95% and 5% respectively. Note that for ease of analysis, the mutation operator was not used. The parent selection scheme is tournament selection.

The population size was set to 500. To prevent code bloat, it is important to limit the size of program trees. Because we investigated the effect of depth of crossover points, we did not use the usual limit of a maximum tree depth, but

instead limited the total number of nodes in trees. We used a limit of 31 nodes[1]. These two parameters are based on empirical search for the given problems. Note that hereafter, unless otherwise noted, the *size* of a program refers to the number of nodes in the program.

We selected parent programs and offspring in the four approximations to the optimal crossover operator based primarily on the fitness values. For competitors having the same fitness value, we preferred the one with a shallower tree depth and a smaller number of nodes.

We constructed five GP systems using Standard Xover and each of the four approximations. Note that when the root of a parent program is a potential crossover point, a GP system using Full Xover will ensure every individual in the next generation is no worse than its parents, so the fitness of the generations is non-decreasing.

The tournament size was set to four when using Standard Xover, based on empirical search. It was set to one when using Full Xover in order to avoid premature convergence caused by parent selection pressure, based on our previous work shown in Chapter 5. We also set the tournament size between one and four for the other three systems according to their different search intensities, respectively one, two, and two for Partial$^+$, Partial, and Partial$^-$, to get the best results for each system.

The actual implementation of tournament selection used in the experiments is the round-replacement tournament selection (see Section 3.6.1 on page 70) instead of the standard tournament selection. This is because although there is no significant difference between the round-replacement strategy and the standard tournament selection for larger tournament sizes, the round-replacement tournament selection has some advantages for smaller tournament sizes.

An evolutionary process is terminated when the number of generations reaches the pre-defined maximum of 51 (including the initial generation), or the problem has been solved (there is a program with a fitness of zero on the training data set), or the error rate on the validation set starts to increase (for BinCla).

We ran experiments using the five GP systems for each of the three problems.

[1]Discussions of a large size limit will be covered in Section 6.9.

For each experiment, we repeated the entire evolutionary process 50 times, independently.

6.5 Effectiveness Comparison

The first step of the analysis compares the problem-solving quality of the five GP systems in order to identify which approximation is closest to the optimal crossover operator. We intentionally ignore the computational cost of fitness evaluation, even though it is clear that the number of evaluations will vary enormously across the different crossover operators, resulting in significantly different computational cost. However, the different crossover operators are merely different simulations of the optimal crossover operator, and the differing computational costs are associated with the differing accuracies of the simulations rather than the actual computational cost if an optimal crossover operator were available. Therefore, the exact computational cost of fitness evaluation is not relevant for this analysis.

Table 6.1 compares the performances of the five GP systems and Table 6.2 lists the number of generations required in each experiment. The measures used here are the same as those described on page 80. Briefly, the measures are the number of failures, the RMS error, and the classification error rate on test data over 50 runs for EvePar, SymReg, and BinCla respectively.

Table 6.1: Performance comparison.

Xover Mode	EvePar Failure	SymReg RMS Error	BinCla Test Error Rate (%)
Standard	50	45.4 ± 6.5	5.2 ± 1.6
Partial$^-$	48	39.1 ± 8.9	4.8 ± 1.5
Partial	19	22.4 ± 10.6	4.7 ± 1.5
Partial$^+$	0	6.5 ± 6.9	4.4 ± 1.3
Full	1	1.6 ± 2.3	4.5 ± 1.5

The tables show that the overall problem-solving quality increases as the search intensity increases, along with the decreased parent selection pressure. For EvePar, systems using Partial$^+$ and Full Xover were approximately equally effective. Almost all runs in the two systems produced an optimal solution, but Full Xover required significantly fewer generations and is therefore the best approximation

Table 6.2: Number of generations required.

Xover Mode	EvePar	SymReg	BinCla
Standard	38 ± 9	45 ± 6	18 ± 13
Partial$^-$	38 ± 8	46 ± 5	16 ± 10
Partial	25 ± 8	40 ± 6	12 ± 9
Partial$^+$	22 ± 5	48 ± 2	13 ± 10
Full	12 ± 2	30 ± 4	7 ± 4

to the optimal crossover operator for EvePar. For SymReg, it is clear that Full Xover outperformed others. For BinCla, systems using Partial$^+$ and Full Xover were as good as or better than the others. In addition, Full Xover required fewer generations and is therefore the best approximation to the optimal crossover operator for BinCla. Note that these performance comparison results assume the appropriate choice of parent selection pressure configuration. Without random parent selection, the greedy one-step search of Full Xover may lead GP search into local optima (see page 147).

Also note that, in the experiments, Full Xover — the ideal constructive crossover operator — happened to be the best approximation to the optimal crossover operator. This may be due to several reasons, including no use of mutation, approximations at coarser levels, random parent selection, and other GP parameter configurations. It would be necessary to do a further investigation to determine the cause.

6.6 DCP Analysis

The next step of the analysis identifies patterns in DCP in the evolutionary process leading to the best solutions. We first extract an approximation to the programs that would have been generated by the optimal crossover operator, and then analyse the depths of the crossover points in the parents of each of these programs.

Our logging system records every generated program along with an ID and, if it is generated by crossover, the IDs of its parents, the position of the crossover point in each of its parents, and the height of each of its parents. At the end of

a run, we locate the best solution found in the run and then trace its ancestry all the way back to the initial population, collecting all its ancestors. This ancestral tree of GPPs (see page 124) represents the best approximation to the evolution that would have been generated by an optimal crossover operator. Note that as the root of a program tree can also be involved, a program can be generated from replacing the entire parent P_1 by the entire P_2 or a strict subtree of P_2. In this case, P_1 will not be considered as a GPP and the corresponding back tracing branch will be terminated. Finally, we analyse the DCP in all the GPPs that were generated by crossover.

Two measures are considered in the DCP analysis. One is called *depth ratio*. The other is the absolute depth of crossover point.

Depth ratio is the height of a crossover point in a parent above the lowest leaf nodes as a fraction of the depth of the lowest leaf node: $(1 - \frac{d}{D})$ where d is the depth of crossover point from the root and D is the maximum depth of the parent tree. Therefore, a large value indicates that the crossover point is close to the root of a tree (so a large fraction of the tree is being substituted), while a small value indicates that the crossover point is close to the bottom of a tree. For instance, if the maximum depths of two parents are 4 and 6 respectively, and the crossover points are at depth 0 and 6 respectively, then the corresponding depth ratios of crossover points are 100% and 0%, reflecting that the crossover point in the first parent is the root node of the whole program tree and that in the second parent is one of the bottom nodes of the whole program tree. Note that programs in the initial population do not have a depth ratio since the initial population is randomly generated.

6.6.1 Depth ratio analysis via boxplot

This section presents visualisations using boxplots of the distribution of depth ratios in a population at each generation along evolution. Figure 6.1 illustrates the distributions of depth ratios by generation along evolution in 50 runs for the five GP systems (five rows) on the three problems (three columns). Thick red bars indicate median values. Red plus signs refer to outliers. The plot starts from generation one.

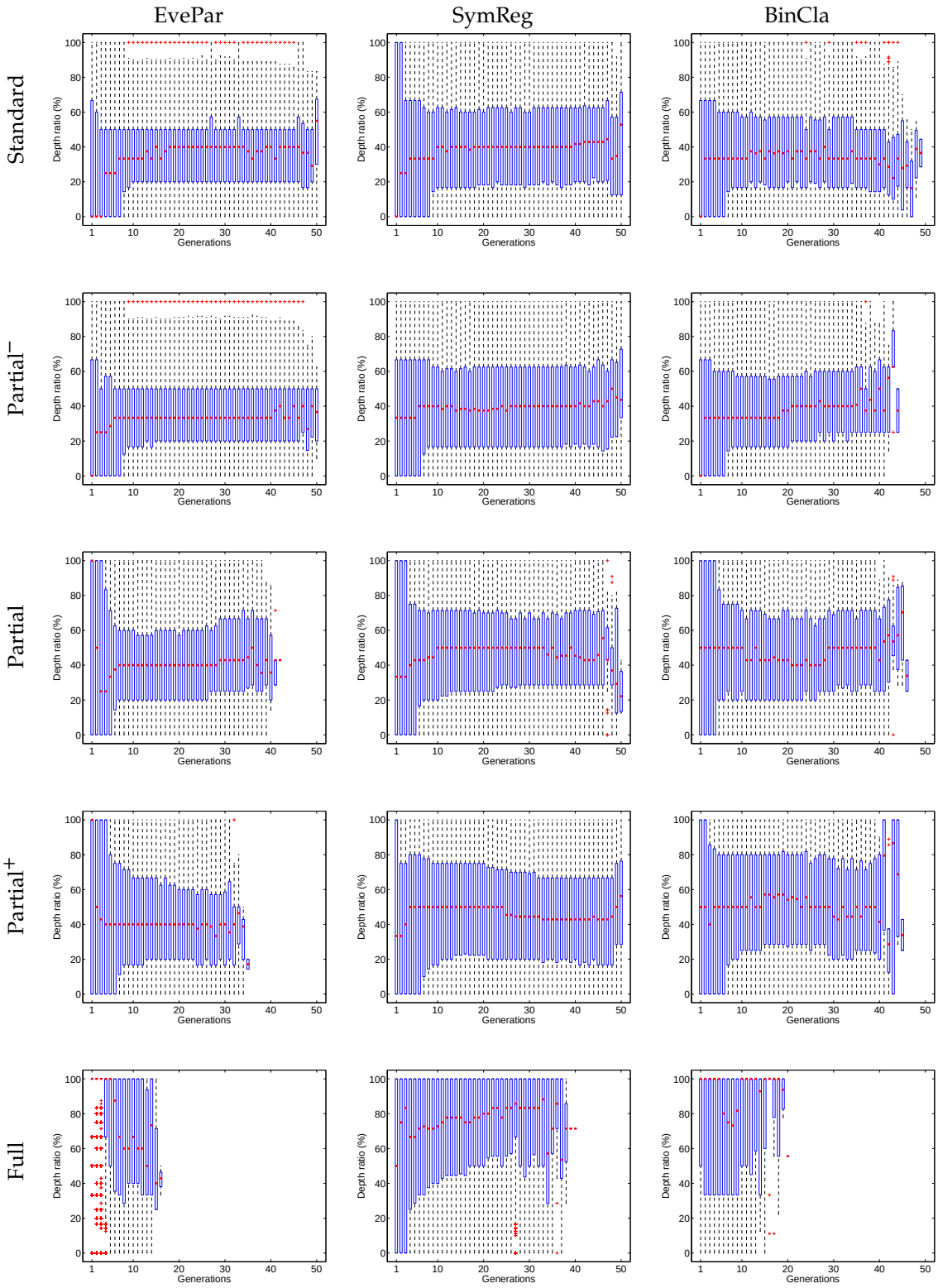


Figure 6.1: Distributions of depth ratios for GPPs involved in crossover along evolution presented in boxplot. The parent size limit is 31 nodes. Thick red bars indicate median values. Red plus signs refer to outliers.

6.6.1.1 Standard and Partial⁻ Xovers

The distribution of depth ratios at a generation in Standard Xover is predictable since the crossover point selection has strong correlation with the program tree shapes in the population. For instance, when creating GPPs in the first generation, the median value at the left-most box of each of the three charts at the first row was zero, suggesting that at least 50% of crossover events occurred at the bottom of their parent program trees for all three problems. This is because although the population initialisation method was the ramped half-and-half method, the maximum depth was set to only four (counted from zero), resulting in limited possible tree variations and a very large number of full trees in the initial population.

When using Standard Xover, an interesting case for SymReg is that the upper quartile of the distribution of depth ratios for producing GPPs at generation 1 reached the roots of parent program trees (i.e. root crossover events), while it was only around 65% for EvePar and BinCla. A possible reason is related to the arities of functions included in the function set, which directly affected the program tree shape. The function sets for EvePar and BinCla both included the three-arity *if* function, binary functions and unary functions, while the function set for SymReg included only four binary functions and three unary functions. By analysing the tree shape of GPPs at the initial populations for SymReg, we found that over 67% of GPPs had depths of only one or two. Half of these depth one GPPs had only one node at depth one and the other half had two nodes at depth one. With choosing crossover points randomly, this would result in at least 25% of crossover events occurring at the roots of parent program trees. The same shape analysis for EvePar and BinCla did not show a similar tree shape distribution as in SymReg. The other interesting finding is that at the middle and later stages of evolution, the bottoms of parent program trees appear to receive little selection preference, which is inconsistent with [64]. Our initial explanation was that this might be because, unlike [64], the experiments did not control code bloat by limiting the maximum program depth, but it will be necessary to conduct further investigation.

When using Partial⁻ Xover, the distribution of depth ratios is very similar to

that of Standard Xover for all the three problems respectively. A possible explanation is that the maximum of 31 nodes in an individual gives an upper bound of three crossover points for each parent program in Partial$^-$ Xover. This small increase of number of crossover point selections appears unable to change the distribution of depth ratios significantly compared with Standard Xover.

6.6.1.2 Partial, Partial$^+$, and Full Xovers

As the search intensity increases significantly, the distributions of depth ratios change noticeably for all the three problems. For Full Xover the distributions are also noticeably different between the three problems.

At the early stage of evolution, the depth ratio is increasingly biased towards the roots of parent programs for EvePar and BinCla as the search intensity increases, and the bias becomes much stronger in Full Xover. For instance, for EvePar almost all crossover points in the early generations are the roots of parent programs, indicating that most of the parents cannot produce any offspring that is better than themselves. Therefore, instead of leading the search to a worse search space by choosing inappropriate crossover points for a given pair of parents, simply retaining the corresponding search space by keeping parents in the next generation seems to be a wise choice. However, the pattern is less clear for SymReg.

At the middle and later stages of evolution, the depth ratio appears to be almost evenly distributed for SymReg and BinCla as the search intensity increases, and the depth ratio is biased towards the roots of parent programs in Full Xover. However, the pattern is less clear for EvePar. These results suggest that an effective depth-control strategy is problem-dependent and is also evolutionary stage-dependent.

Since Partial and Partial$^+$ Xovers are able to improve the GP search performance noticeably for SymReg, the corresponding evenly-distributed depth ratios at the middle stage of evolution appear to be evidence supporting the assignment of equal probabilities to different depths.

Furthermore, we found that the distributions of depth ratios in the first four GP systems are very similar to each other for SymReg and BinCla. However,

the different performance changes in the four GP systems for SymReg and BinCla provided an interesting finding. For SymReg, large performance changes occurred in the four GP systems, indicating that when selecting crossover points, a slight change in average depth may be associated with a remarkable improvement. In contrast to SymReg, the differences in performance between the four GP systems for BinCla are not as significant, while the depth ratio distributions have more variations in BinCla than in SymReg. This observation suggests that sometimes there may be only slight performance improvement associated with changes in average depth. To explore possible reasons, problem difficulty may need to be taken into account. BinCla, on the one hand, seems to be a simple problem as about 95% test accuracy can be reached by the GP system using Standard Xover. On the other hand, it seems to be very hard to do significantly better than 95% because Full Xover can obtain only less than 1% improvement after putting in a large amount of effort.

In summary, the results of depth ratio analysis via boxplot demonstrate that distributions of crossover points that give higher weight to nodes close to the root of the tree than Standard Xover are associated with better performance but it is not straightforward to develop the most effective depth-control strategy, which is clearly problem-dependent and evolutionary stage-dependent. We also do not know how much the intensive search strategies contribute to the performance and whether depth control by itself is sufficient.

6.6.2 Issues of boxplot analysis

The above analyses via boxplot may have two issues. One issue is related to the boxplot technique itself. Boxplot as a visualisation technique sometimes does not provide sufficiently detailed information. It has a many-to-one mapping problem. For instance, three different distributions shown in a histogram can have the same boxplot results (see Figure 6.2). It is not clear whether similar cases occurred in our experiments, leading to some misinterpretations.

The other issue is related to the way depth ratios were plotted against generations. We plotted depth ratios against this number of generations since the first

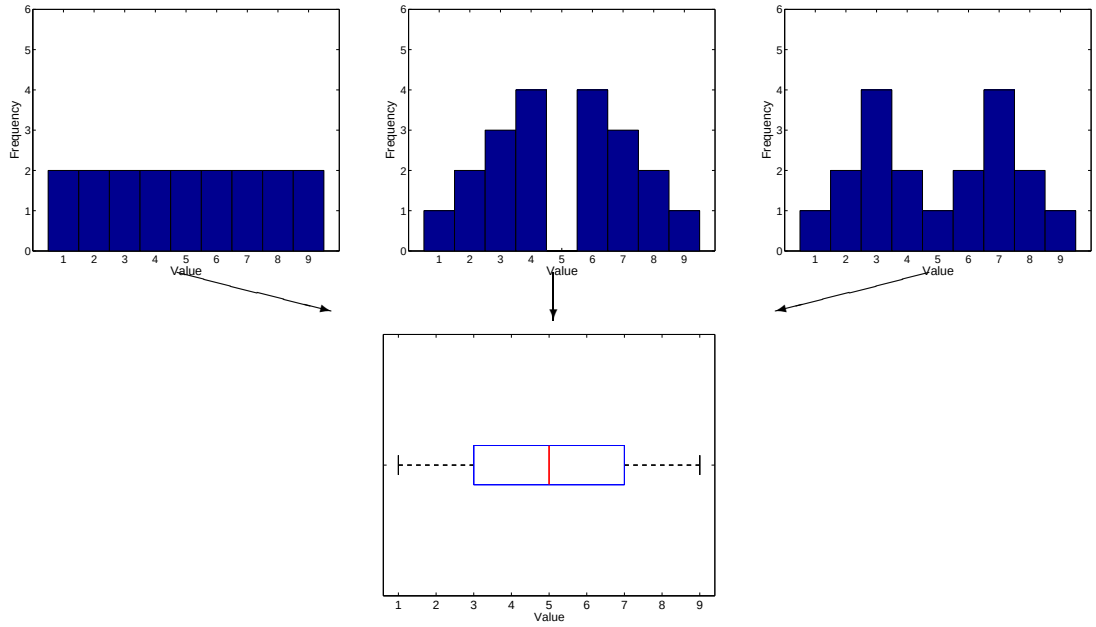


Figure 6.2: Different distributions can have the same boxplot result.

generation, which should work well in general. However, as the best solution in a run may appear at any generation, *the last generation*, where the best-of-run solution appears first time, varies from run to run. Furthermore, there are very few GPPs in the last several generations of any run — only one GPP at the last generation and no more than 16 GPPs at the fifth-to-last generation in a run. Therefore, the data for generations at the right end of the plots may be based on very few GPPs.

Therefore, in order to expose more details of the distributions, we visualised depth ratios using another technique we called *grayplot* and also plotted the results against the number of generations before the last generation.

6.6.3 Depth ratio analysis via grayplot

The grayplot is a kind of histogram in which the frequencies are represented by areas of different gray levels. Since we are particularly interested in the frequencies of depth ratios of 0% and 100%, we used a histogram with non-uniform bin sizes to record the frequencies for depth ratios at each generation along evolution over 50 runs.

We used 12 bins. The first bin counts the depth ratio of exactly 0% and the last bin counts the depth ratio of exactly 100%. The second bin counts the depth ratios

between $(0\%, 10\%]$, the third bin counts the depth ratios between $(10\%, 20\%]$, and so on, and the 11th bin counts the depth ratios between $(90\%, 100\%)$. For each generation, we normalised the frequency in each bin by the sum of frequencies in all bins at the same generation, thus the normalised frequency in each bin is within $[0, 1]$. It is likely that the number of bins is not optimal, and we could follow [17] to develop an optimisation method for choosing the number of bins. This, however, is beyond the scope of this thesis.

Different gray levels map to different normalised frequencies, where black maps to the normalised frequency one and white maps to the normalised frequency zero. Therefore, the darker the area, the higher the frequency. Note that normalising frequencies in this way means that care should be taken when interpreting gray levels at a generation where the total number of data samples is relatively small.

Figure 6.3 illustrates the distributions of depth ratios for each generation, while Figure 6.4 illustrates the distributions also for each generation but counted back from the last generation. Note that it is not necessary for the index of the last generation to be the same as the predefined maximum number of generations. However, for easy reading, we deliberately adjusted the index of the last generation in a run to be the predefined maximum generation so that the rightmost column in each chart always has 50 data samples (as there is one GPP at the last generation in each of the 50 runs).

As a result, a suitable way to interpret the information presented in these two figures is to read the left half of each chart in Figure 6.3 for the early and middle stages of evolution and the right half of each chart in Figure 6.4 for the middle and later stages of evolution.

The rest of this section partitions the five GP systems into two groups and analyses the changes of the distributions of depth ratios for all three problems in each group.

6.6.3.1 GP systems using Standard, Partial$^-$, Partial, and Partial$^+$ Xovers

The first group consists of the four GP systems using Standard, Partial$^-$, Partial, and Partial$^+$ Xovers. For the four GP systems, information of the distributions

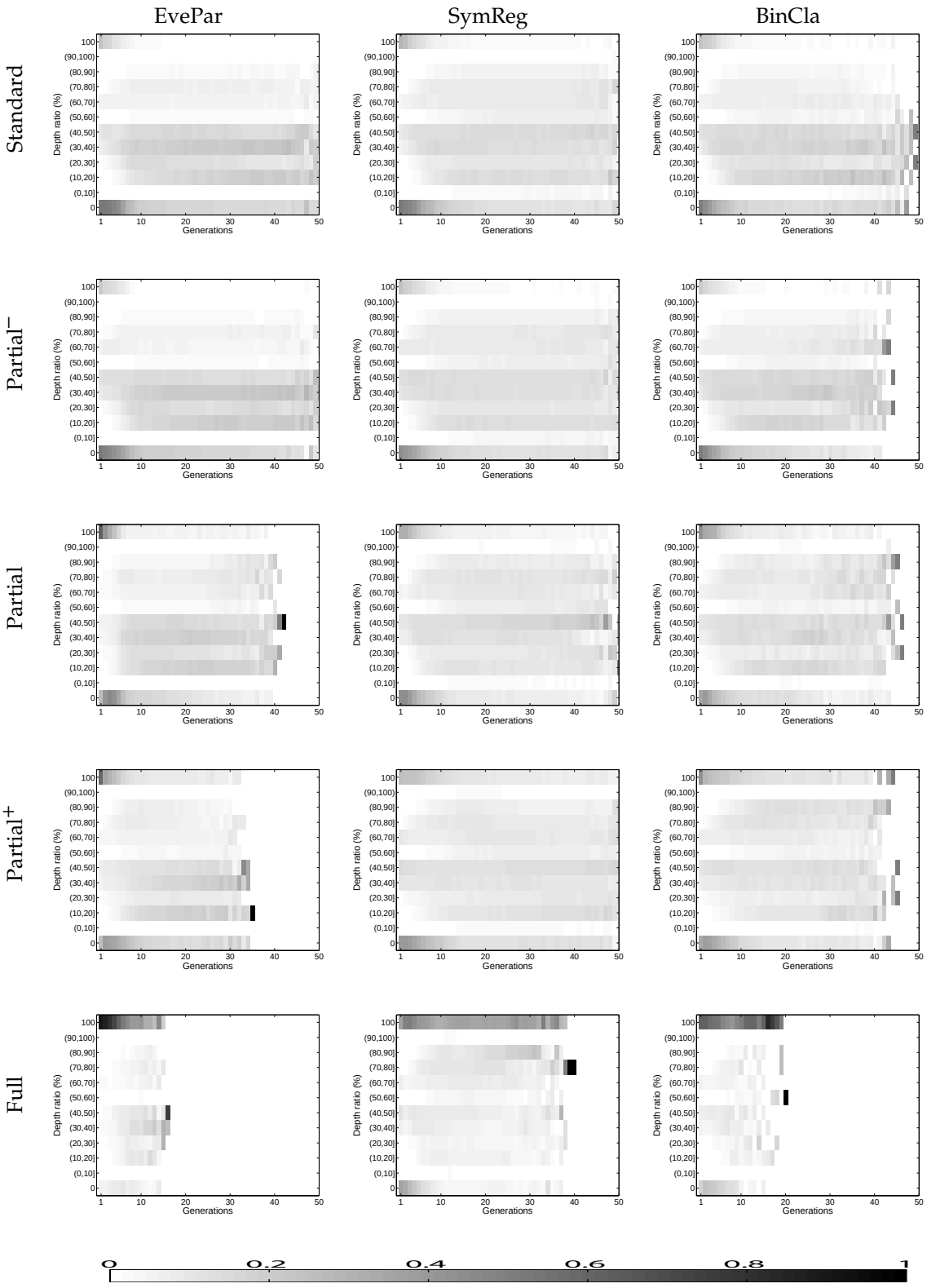


Figure 6.3: Distributions of depth ratios for GPPs involved in crossover along evolution presented in grayplot and normalised within each generation. The parent size limit is 31 nodes. *Read the left half of each chart for the early and middle stages.*

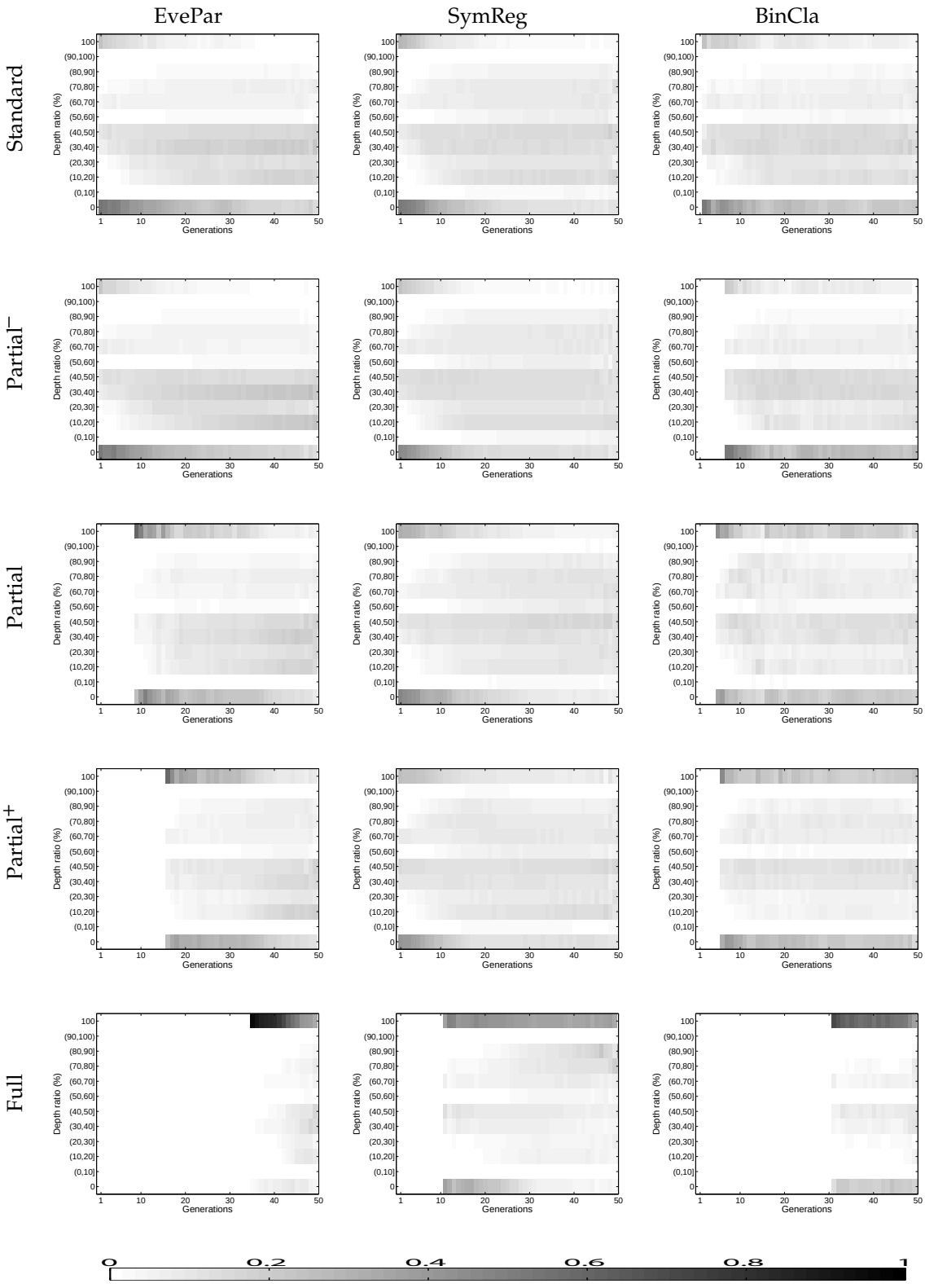


Figure 6.4: Distributions of depth ratios for GPPs involved in crossover along evolution presented in grayplot and normalised within each generation. The parent size limit is 31 nodes. *The plot is against the number of generations before the last generation. Read the right half of each chart for middle and later stages.*

presented in grayplots is consistent with that presented in the boxplots but clarifies it in several important ways:

- For Standard Xover, the bottoms of parent program trees did receive noticeable selection at a similar degree to other low depth ratios at the middle and later stages of evolution for all the three problems, although the depth-limiting code bloat control strategy was not used.
- The depth ratios were almost evenly distributed at the middle and later stages of evolution when using Partial and Partial$^+$ Xovers for SymReg but not for BinCla (as in boxplots).

Furthermore, from Figures 6.3 and 6.4, in the middle and later stages of evolution, the selection preference changes with increasing search intensity from mainly below the depth ratios of 50% to a wider range but with different preferences. However, the depth ratios in the $(30\%, 50\%]$ range still received higher preference than other ranges.

6.6.3.2 GP system using Full Xover

The distributions of depth ratios in the GP system using Full Xover significantly differ from those in the other GP systems. The grayplots provide more precise information about the distributions than the boxplots.

It is clear that the roots of parent program trees consistently received much higher selection preference than nodes at other depths across the whole evolutionary process for all the three problems.

However, from the last row of charts in Figure 6.4, the distributions of depth ratios (except the depth ratio of 100%) in the middle and later stages of evolution are different between the three problems. For EvePar, depth ratios in the $(30\%, 50\%]$ range received higher selection preference. For BinCla, in addition to the depth ratios in the $(30\%, 50\%]$ range, the bottoms of parent program trees also received higher selection preference. For SymReg, depth ratios in the $(70\%, 90\%]$ range received higher selection preference. According to the ways of calculating and partitioning depth ratios, the pattern suggests that GPPs in SymReg may be

much deeper than those in EvePar and BinCla. To verify the finding, it is necessary to conduct further analysis of absolute depth of crossover points.

In summary, not only did the depth ratio analysis via grayplots support the findings obtained via boxplots, it also provided more details of depth ratio preferences than the boxplots.

6.6.4 Absolute depth of crossover point analysis via grayplot

Note that in the Figures 6.3 and 6.4, the depth ratios in the $(0\%, 10\%]$, $(50\%, 60\%]$, and $(90\%, 100\%)$ ranges have a very low frequency. This is probably an artifact of the way the histogram bins were chosen. For example, if the height of all GPPs at a generation is less than 10, then the depth ratios will never be in the $(0\%, 10\%)$ or $(90\%, 100\%)$ ranges, regardless of where crossover points are. In addition, GPPs at different evolutionary stages for different problems have different heights that influence depth ratios. Therefore, further analysis of absolute depth of crossover point is required to clarify the findings. This section visualises and analyses the distributions of absolute depth of crossover points via grayplot.

For each experiment, we plotted distributions of DCPs (across all generations) against the depth of parent GPPs in grayplot. Figure 6.5 illustrates the distributions in the five GP systems for each of the three problems. In each chart, the x-axis is the depth of the parent program tree. The y-axis is the absolute depth of crossover point where depth zero is deliberately set at the top in order to match the plots in Figures 6.1, 6.3 and 6.4. The top horizontal line of data in the figure represents the root crossover events and the diagonal line of data (where $x = y$) represents crossover events involving the bottoms of parent program trees. Since the median value tends to be less affected by outliers than the mean, Figure 6.5 also shows median values in linked squares to assist the visualisation.

It is clear that when a parent has zero depth (a single node program), the absolute depth of crossover point must be zero as well. Therefore, the small square at the topmost left corner in each chart is 100% dark. It is also clear that across the five GP systems the parent program depths in SymReg are larger than those in EvePar and BinCla. This might be because 75% of functions in SymReg are unary functions; thus with the same number of nodes, trees in SymReg may be

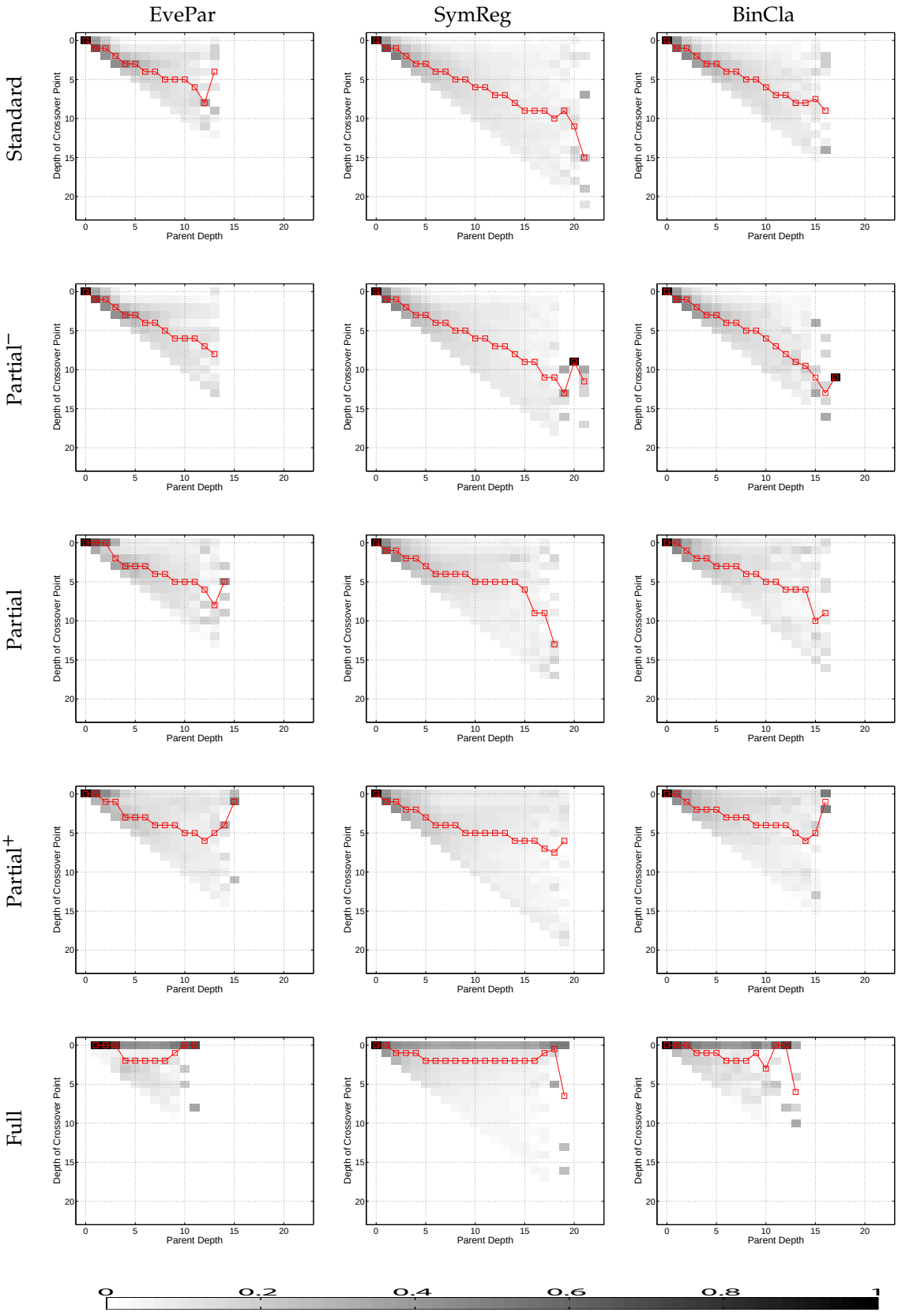


Figure 6.5: Distributions of absolute depths of crossover points for GPPs involved in crossover partitioned by parent depth and rescaled within each partition. Median values are highlighted by linked squares. The parent size limit is 31 nodes and the maximum depth which appeared in the experiments is 21.

sparser and deeper than those in EvePar and BinCla. This observation confirmed the explanation of Full Xover having a higher preference of the $(70\%, 90\%]$ range in SymReg than EvePar and BinCla.

In the GP systems using Standard Xover for all the three problems (the first row of charts in the figure), the bottom of shallow parent programs (less than five depths) received stronger selection preference than deep parents. Recall that we found the bottoms of parent program trees received noticeable selection preference at the middle and later stages of evolution from the DCP analysis via gray-plot. This observation may suggest that the fraction of shallow parent programs was larger than deep parents at the middle and later stages of evolution. Also, as the depths of parent programs increase, the range of the absolute depths of crossover points starts to spread out randomly, shown by a wide area with similar gray levels on and above the diagonal line in the charts, but with a bias towards the bottoms of parent program trees, shown by the median value line being close to the diagonal line in the charts. This pattern might suggest that GPPs of the same depth may have different tree shapes but the number of nodes close to the bottoms of program trees are larger than that at any one of the other depths.

As the search intensity increases, especially when Full Xover is used, the absolute depths of crossover points become biased towards the roots of parent program trees. This is shown by 1) the gray levels at the top horizontal line being much darker than those at other areas, and 2) the median value line being close to and even overlapping the top horizontal line of data in the charts. This pattern applies to all three problems with some slight variations, which may be caused by not having enough sample data in some groups of GPPs. For instance, in the Full-Xover-BinCla chart, the number of GPPs of depth 13 is only six, of which two GPPs have crossover points at depth zero, two GPPs have crossover points at depth 10, one GPP has a crossover point at depth four, and another GPP has a crossover point at depth eight, resulting in the median value of six.

The patterns of absolute depth of crossover point presented in this figure together with the patterns of depth ratios obtained in previous sections suggest that:

- An effective depth-control strategy is problem-dependent and evolutionary

stage-dependent. Applying the same control scheme for any problem and through the whole evolutionary process would not provide the most effective GP system. As tree shape is a consequence of different problems and different evolutionary stages, it should be taken into account when developing an effective depth-control strategy.

- Crossover point selection preference is strongly biased to roots of parent trees. This suggests that if parents could not find any better offspring after several tries, simply choosing the root as the crossover point to effectively retain parents in the search space may be more beneficial than selecting an inappropriate crossover point which may lead the search to a worse space. An appropriate offspring search intensity is worth further investigation.

- Crossover point selection preference is also biased to the bottoms of parent program trees but less strongly than roots, and the preference becomes biased to some ranges of depths between root and bottom as the depth of the parent trees increases. This suggests that unequal-depth-selection-probability strategies with a consideration of parent tree depth is a better practice than equal-depth-selection probability strategies.

6.7 SSS Analysis

The previous section analysed the depth of crossover point. Except for the strong preference for root nodes, it is not clear whether depth control by itself is sufficient because the size of the subtree rooted at a crossover point at a given depth could vary. Knowing distributions of the sizes of the substituted subtrees could be useful for developing robust heuristics for selecting good crossover points. This section conducts the analysis of substituted subtree sizes (SSS).

The section applies the same methodology used in the DCP analyses to the SSS analyses based on two measures: subtree size ratio and absolute subtree size.

6.7.1 Subtree size ratio analysis

The subtree size ratio is the number of nodes in the subtree rooted at a crossover point in a parent as a fraction of the total number of nodes of the parent. According to the results of the DCP analysis, the frequencies of single-node subtrees and whole program trees are worthy of attention in the SSS analysis. To make it easier to identify single-node subtrees, we subtracted 1 from both the subtree and parent sizes so that a single-node subtree always resulted in a subtree size ratio of 0%. For the special case of a single-node parent, the ratio is defined to be 100%.

Figure 6.6 illustrates the distributions of subtree size ratios by generation along evolution in 50 runs for the five GP systems (five rows) on the three problems (three columns). The plots start from generation one. Figure 6.7 shows the distributions against the number of generations before the last generation (the generation where the best-of-run solution appears first time).

6.7.1.1 GP system using Standard Xover

For Standard Xover, it is clear that leaf nodes have the highest selection preference along evolution for all the three problems. Another interesting pattern in Standard Xover is that there are some selection preference for the root nodes but only in the early stage of evolution for all the three problems. This pattern may be due to GPPs of shallow and sparse tree shapes, which give the roots relatively high selection probabilities. These appeared relatively often in the early stage of evolution but seldom after that.

To verify the hypothesis, we conducted a simple program tree shape analysis. In each generation, we firstly grouped GPPs based on their maximum depth, then partitioned the GPPs in each group by their sizes and calculated their frequencies for each size. As Standard Xover chooses crossover point uniformly, we can calculate the probability that the roots of GPPs are chosen as crossover points based on these frequencies and the sizes of GPPs.

The analysis shows that there were very few GPPs whose maximum depth is just one or two from generation 8 for all the three problems. The result indicates that program trees become deep and bushy when the evolutionary process moves

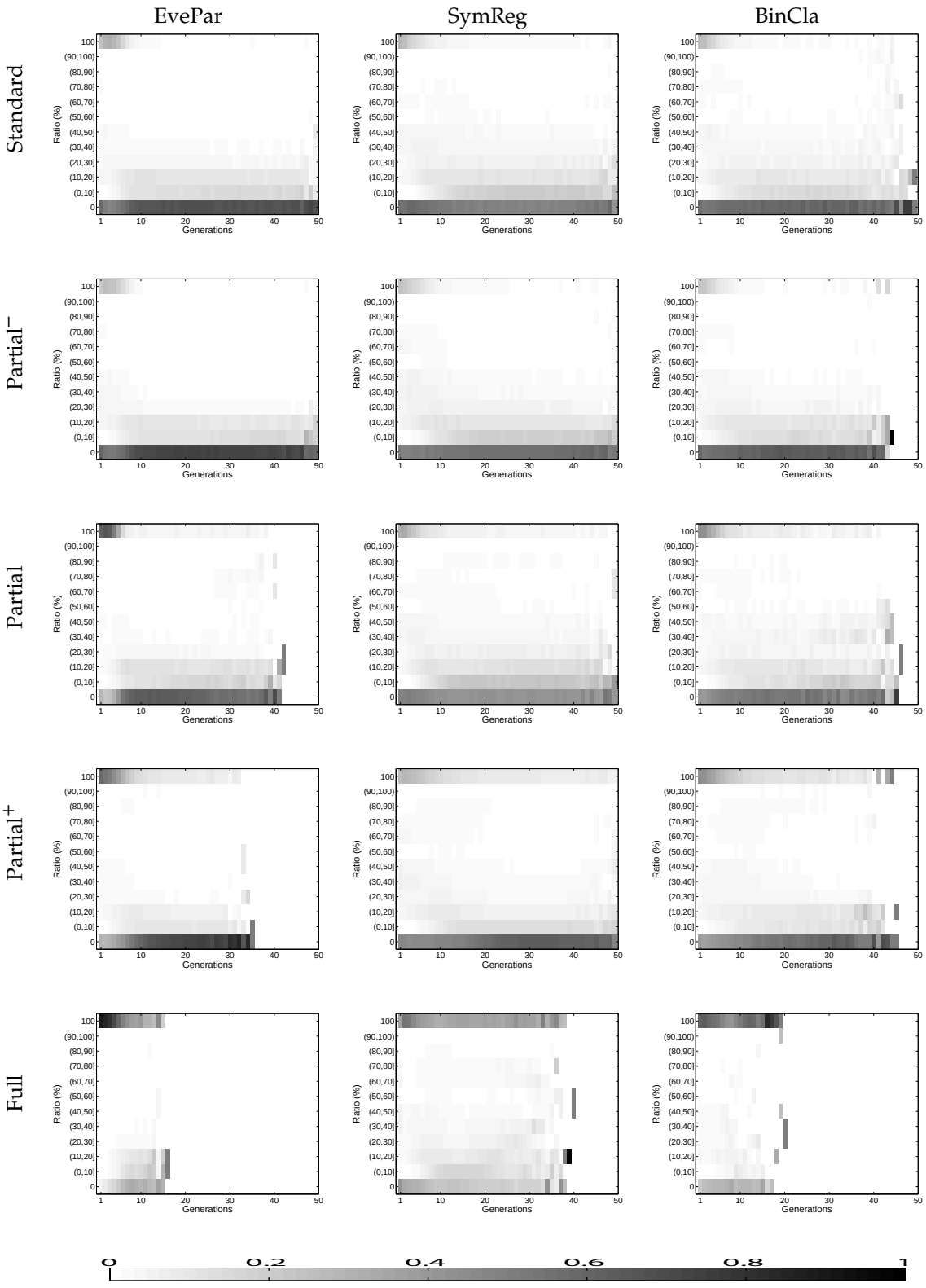


Figure 6.6: Distributions of subtree size ratios for GPPs involved in crossover along evolution presented in grayplot and normalised within each generation. The parent size limit is 31 nodes. *Read the left half of each chart for the early and middle stages.*

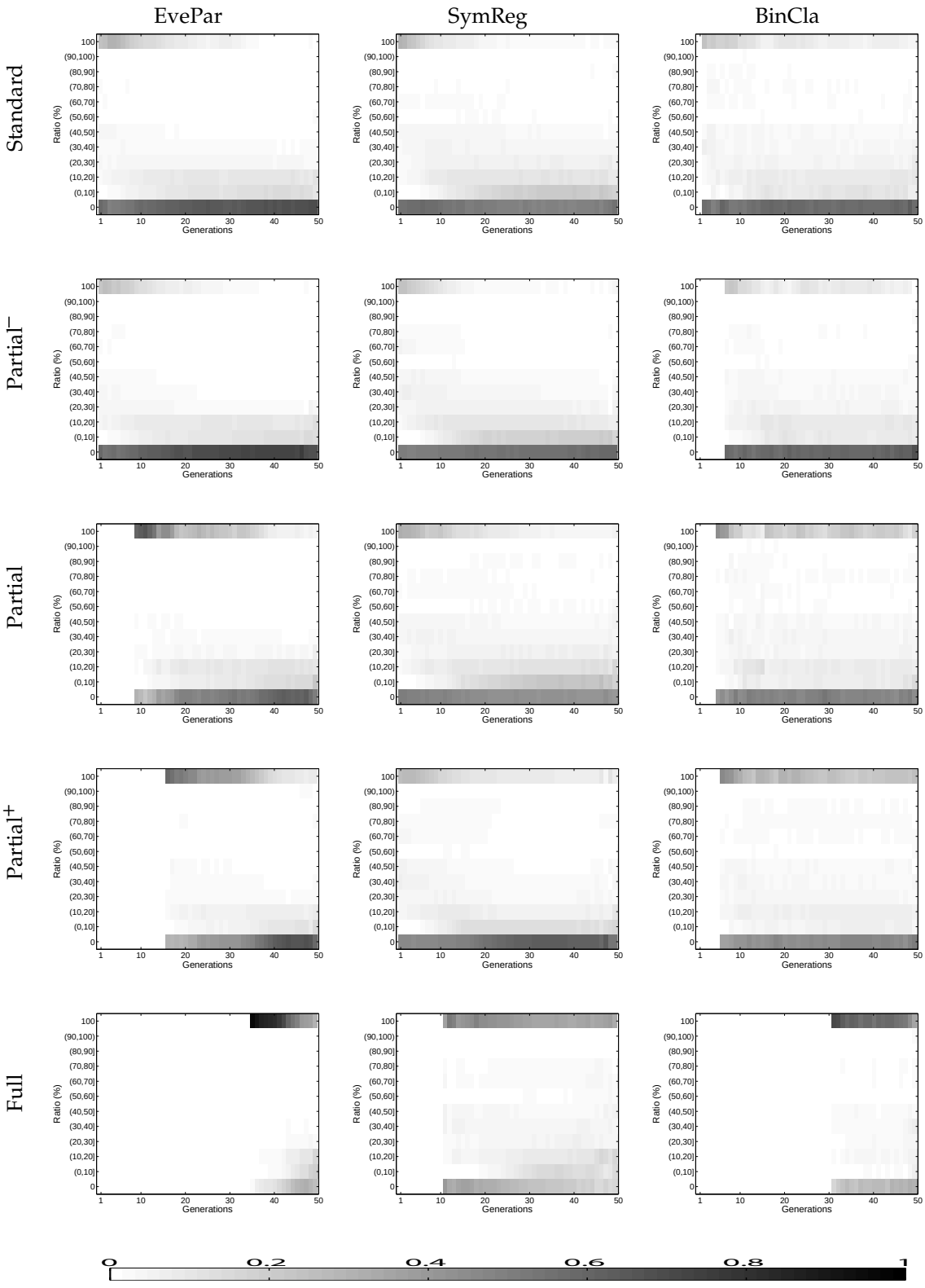


Figure 6.7: Distributions of subtree size ratios for GPPs involved in crossover along evolution presented in grayplot and normalised within each generation. The parent size limit is 31 nodes. *The plot is against the number of generations before the last generation. Read the right half of each chart for middle and later stages.*

on, resulting in the roots of program trees having a very low probability of being selected as crossover points. This simple program tree shape analysis verified the pattern found in Standard Xover.

As the search intensity increases, the roots of GPPs receive a higher selection preference, not only in the early stage of evolution, but also extended to middle and even later stages of evolution. In contrast, the selection preference for leaf nodes decreases. Since we are particularly interested in the pattern in Full Xover, the next section skips Partial$^-$, Partial, and Partial$^+$ Xovers and focuses only on Full Xover.

6.7.1.2 GP system using Full Xover

For Full Xover, in EvePar, the subtree size ratios during the early stage of evolution are almost 100%, meaning the roots of GPPs (including single-node GPPs) are selected most of the time. This is consistent with the findings obtained in the DCP analysis. In addition, close to 50% of substituted subtrees are leaf nodes in the later stage of evolution. In Figure 6.4, the non-root nodes were spread out among the depth ratios from 0% to 50%; Figure 6.7 makes it clear that they were almost all leaf nodes, though at different depths in the tree.

In SymReg, the substituted subtrees during the early stage of evolution are mainly entire program trees or single-node subtrees with a slight bias to single-node subtrees. In the middle and later stages of evolution, the pattern continues though less strongly as small non-root and non-leaf subtrees raise to about 20%. By examining the corresponding charts in Figures 6.3 and 6.4, we can see that these small subtrees may appear at any depth.

In BinCla, the substituted subtrees during the early stage of evolution are also mainly the entire program trees or single-node subtrees but with a clear bias to the entire program trees and the pattern remains in the rest of evolution. By examining the corresponding charts in Figures 6.3 and 6.4, we can see that some of these leaf nodes appear around the middle depth of parent program trees.

The patterns discovered in the above SSS analyses suggests that the distributions of substituted subtree size ratios are more consistent across the problems than the distributions of depth ratios, but are still somewhat evolutionary stage-

dependent. When designing a good crossover point selection strategy, combining both subtree size and depth should be able to provide better performance than using either alone.

6.7.2 Absolute subtree size analysis via grayplot

Figure 6.8 illustrates the distributions of absolute subtree size for GPPs involved in crossover in the five GP systems for each of the three problems. In each chart, the x-axis is the size of the parent program tree. The y-axis is the absolute subtree size. Therefore, the bottom horizontal line of data in the figure represents the leaf nodes being selected as crossover points and the diagonal line of data in the figure represents the roots of parent program trees being selected.

Standard Xover randomly selects subtrees for crossover; the result is that the distribution of SSS is governed merely by the shape of parent program trees. As expected, there is a strong preference for small subtrees, especially subtrees of size one: since the number of leaf nodes is generally about 50% of the total number of nodes in a program tree, leaf nodes have a higher probability of being selected [145]. Also as expected, there is a preference for smaller GPPs among the root nodes (the diagonal line of data in the figure) because the root is more likely to be selected in small trees.

As the search intensity increases and fitness is taken into account, the distribution changes. Small subtree sizes are still common; however the proportion of root crossover events increases. In the last row where Full Xover is used, the preference for roots is much higher than in other rows. The median values are no longer stuck on the subtree size of one but move towards the roots.

6.8 Analysis of Root Crossover Events

The roots of program trees were very frequently used for crossover by Full Xover. Because the three different kinds of root crossover events (root-root, root-in, and root-out) represent very different kinds of modifications to program, it is important to look at this category more carefully: root-root events are effectively copying one of the parents; root-in events insert a whole program into the other

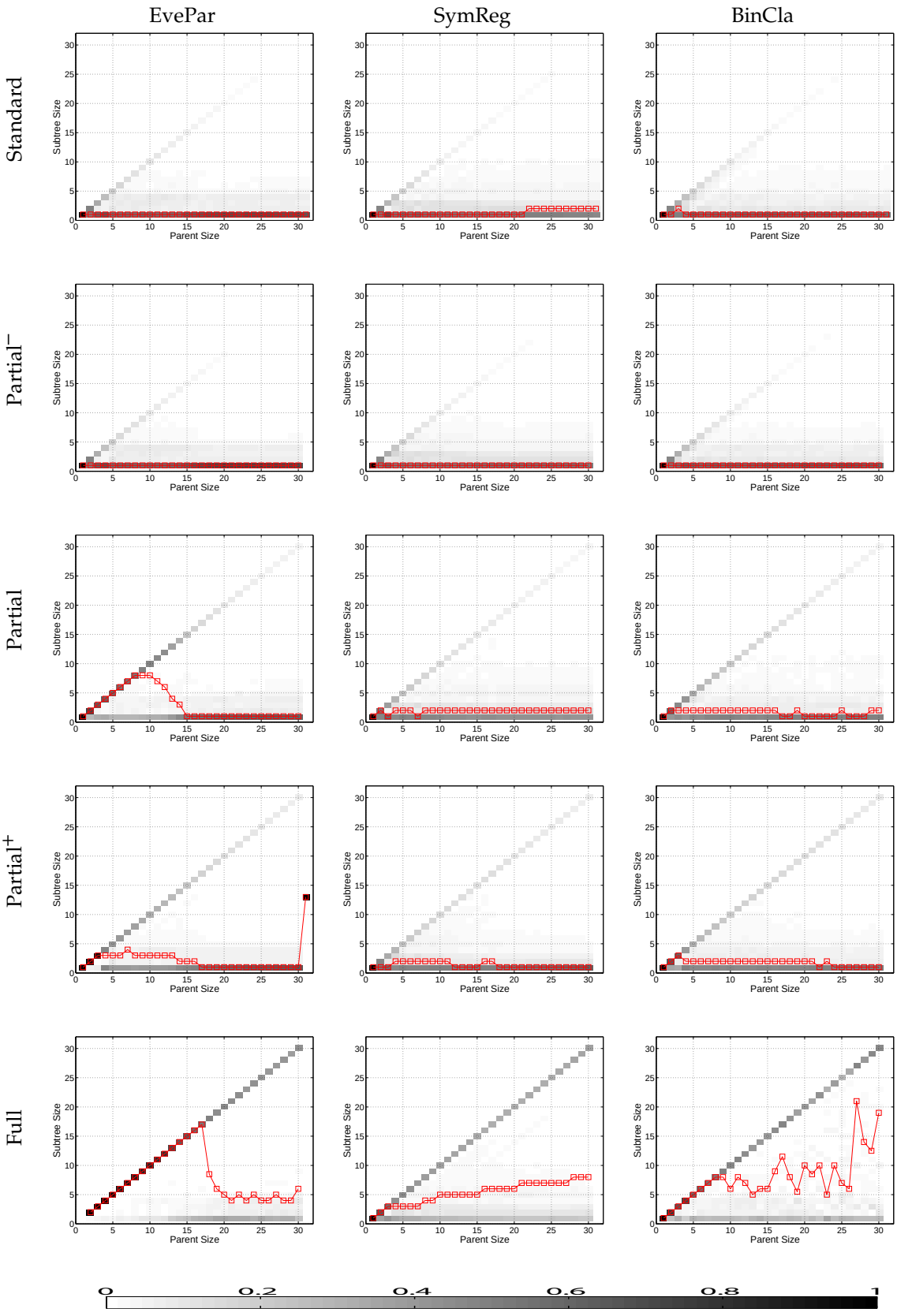


Figure 6.8: Distributions of absolute subtree sizes for GPPs involved in crossover partitioned by parent size and rescaled within each partition. Median values are highlighted by linked squares. The parent size limit is 31 nodes.

program; and root-out events extract a subtree as a new program.

Table 6.3 shows the average ratio of the number of each type of root crossover events to the number of all crossover events involving GPPs, across 50 runs. This is presented for Standard Xover and Full Xover on the three problems.

The relatively low ratio of root-out operations in all cases suggests that subtrees alone seldom outperform their parents[2].

Table 6.3: Average ratio of the number of each type of root crossover events to the number of all crossover events involving GPPs for Standard Xover and Full Xover.

Xover Mode	Xover Type	EvePar	SymReg	BinCla
standard	root-root	2.4 ± 1.2	2.1 ± 0.4	4.5 ± 4.1
	root-in	7.9 ± 3.0	5.4 ± 0.8	17.3 ± 16.3
	root-out	4.3 ± 2.1	1.8 ± 0.4	2.9 ± 3.1
full	root-root	66.6 ± 3.0	31.1 ± 1.5	34.7 ± 16.6
	root-in	14.9 ± 2.1	12.2 ± 2.6	21.6 ± 14.2
	root-out	0.1 ± 0.3	2.0 ± 0.4	4.0 ± 3.8

From the table, when Standard Xover is used, there are not many copy-like crossover operations. This is unsurprising due to its uniform random selection of crossover points: the probability of the root of a parent being selected is relatively low and the probability of the roots of two parents both being selected is even lower. Note that for BinCla, since there was a larger proportion of GPPs of size two, the ratios of root-root and root-in operations are both higher than that for EvePar and SymReg.

In Full Xover, where the crossover operator is deterministic, a large proportion of crossover operations are the same as copy operations. Nordin *et al.* demonstrated that most crossover events in Standard Xover produce offspring with less than half of the fitness of their parents [133, 134]. Our analyses show that even using the best approximation of the optimal crossover operator, there are, on average, 67%, 31%, and 35% crossover events involving GPPs in EvePar, SymReg, and BinCla respectively, in which no possible offspring are better than the parents[3]. This result demonstrates that the ability of standard GP crossover to gen-

[2]This result also suggests that promotion mutation [163] might have advantages in some cases but its effectiveness would be uncertain because in general its execution is controlled randomly, not deterministically.

[3]Note, this assumes the use of a size-limiting bloat control strategy.

erate good offspring is far below what was expected.

Furthermore, the relatively large proportions of root-in crossover events for the three problems show that good offspring can often be produced by using one parent program to replace a subtree of the other. This would suggest that the common technique of not including the root as a potential crossover point is counter-productive. As mutation operators were not used in the experiments, a further investigation is necessary to verify the finding.

6.8.1 Investigating the effect of high copy rate

It might be argued that similar search performance could be achieved by increasing the copy rate deliberately while decreasing the crossover rate. We conducted experiments to test this. In order to determine what the copy rate should be, it was necessary to explore how the three types of root crossover events and the normal crossover event are distributed in different-sized parent programs.

Figures 6.9 and 6.10 illustrate the distribution of the three types of root crossover events and the normal subtree-subtree crossover event in terms of absolute occurrences and relative ratios over the 31 different-sized parent programs respectively for Standard Xover and Full Xover.

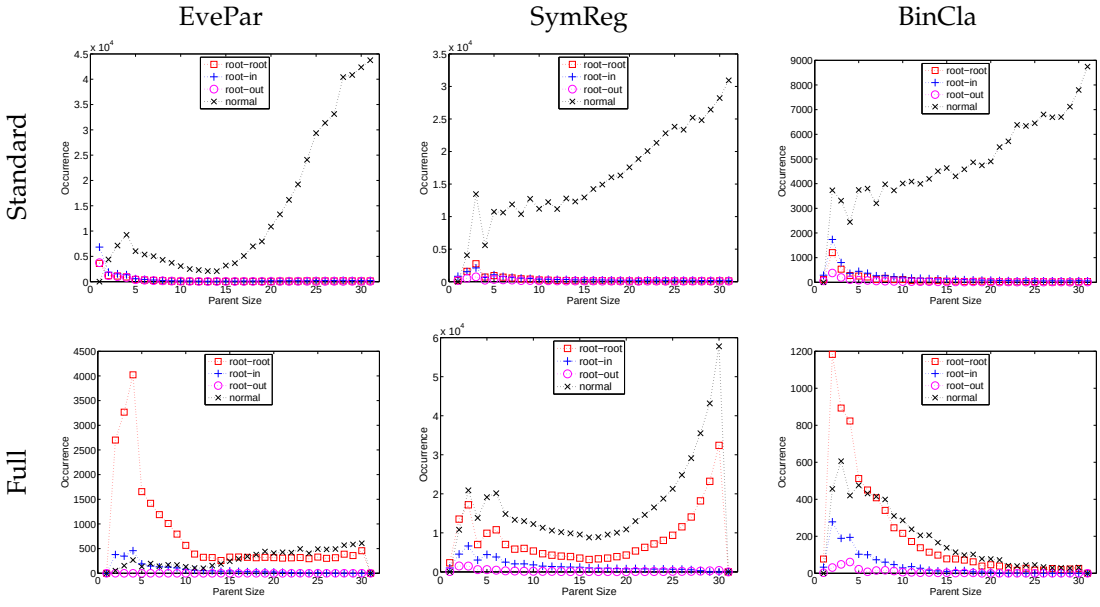


Figure 6.9: Distribution of absolute occurrences of the three types of root crossover events and the normal crossover event (subtree-subtree) in GPPs for Standard Xover and Full Xover.

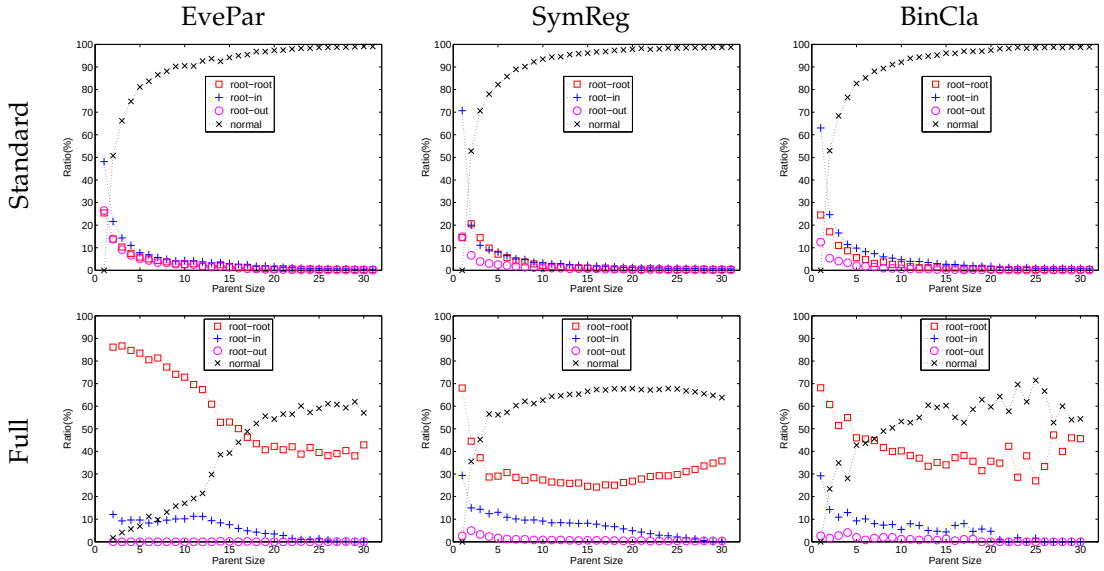


Figure 6.10: Distribution of the relative ratios of the three types of root crossover events and the normal crossover event (subtree-subtree) in GPPs to total crossover events in GPPs for Standard Xover and Full Xover.

The figures show that the absolute occurrences of the three types of root crossover events and the normal crossover event vary with different-sized parent programs, but the relative ratios do show some patterns. It appears that when Standard Xover is used, the distributions are very similar for different problems. The percentage of root crossover events is not large. Most of the root crossover events occur on smaller-sized parent programs. This is understandable as the smaller size the parent is, the higher probability the root is chosen. The distributions are theoretically predictable since in a program the probability of a node being chosen as a crossover point is just a function of the program size.

When Full Xover is used, the copy-like crossover operations are significantly biased to smaller-sized parent programs but also occur very often, about 30% to 40%, on larger-sized parent programs. EvePar's distribution is quite smooth. SymReg's distribution is mostly constant. BinCla's distribution fluctuates between parents of sizes in the range from 20 to 30 but the fluctuation is around 40%. Another interesting finding is that most of the root-in crossover events involve parent programs of sizes less than 20. This might be a consequence of the size-limiting bloat control method, which prevented crossover from using programs of large sizes. However, it might be also because these small-sized parents are probably good high-level features or building blocks. It would be very inter-

Table 6.4: Modified parameters in HighCopy-Full.

	EvePar	SymReg	BinCla
Crossover Rate	55%	65%	65%
Copy Rate	45%	35%	35%

esting to investigate the underlying reasons further.

When designing further investigations of the utility of increasing the copy rate, we took into consideration the results of the previous chapter, which showed that the parent selection pressure should be turned off when applying high offspring selection pressure (i.e. applying the Full Xover) in order to avoid being confined in local optima. Therefore, our further investigation used a baseline of the GP system using Full Xover with the low copy rate (LowCopy-Full) and a tournament size of one. We compared this to another GP system *HighCopy-Full*, which had a higher copy rate and a lower crossover rate. The parameters were modified according to the approximate lower bound of the portions of the copy-like crossover events, namely 40%, 30% and 30% for EvePar, SymReg, and BinCla respectively. Details are shown in Table 6.4.

We conducted the same experiments on the three problems using HighCopy-Full. Table 6.5 shows its performance measures and the average number of generations required.

Table 6.5: Performance measure and generations required in HighCopy-Full.

	EvePar	SymReg	BinCla
Performance	4	4.5 ± 4.8	4.6 ± 1.5
Generations Required	14 ± 2	28 ± 5	7 ± 4

From the table, the performance measure of HighCopy-Full was slightly worse than that of LowCopy-Full for EvePar and SymReg but was similar to that for BinCla (compared to Table 6.1). The numbers of generations required in HighCopy-Full were similar to that in LowCopy-Full (compared to Table 6.2). We also visualised the distributions of absolute subtree sizes of GPPs and the distributions of the three types of root crossover events and the normal crossover event in HighCopy-Full and observed that they were very similar to that in LowCopy-Full respectively.

The results showed that HighCopy-Full could provide comparable performance to LowCopy-Full but was more efficient than LowCopy-Full. By reducing the crossover rate, some of the time-consuming copy-like crossover operations were replaced with low-cost copy operations. However, some other crossover operations were also replaced with copy operations due to random decisions on performing crossover on chosen parents. In order to effectively and efficiently determine whether none of the offspring is better than their parents so that a direct copy operation can be performed, other directions should be taken, such as examining the performance profiles of chosen parents before crossing them over.

6.9 Discussion of Impact of Size Limiting on Crossover Point Selection

In our experiments, the crossover point selections are affected by the size-limit method used for controlling the code bloat issue in GP. When parent sizes are close to even half the predefined size-limit, the position of the node selected in one parent may constrain the positions of possible crossover points in the other.

Although Full Xover is supposed to explore all possible crossover points in a given pair of parents, the actual number of crossover points considered will generally be smaller. In contrast, the impact of the size-limit method on the crossover point selection in Standard Xover should be smaller since the probability of choosing a node close to the root of a parent tree is low. Therefore, the observations and findings obtained from the experiments with the size-limit of 31 nodes are artificially biased, and the bias increases as the search intensity increases. The question is whether this bias is significant.

To answer this question and to verify the outcomes obtained, we conducted additional sets of experiments without using the size-limit. Certainly, the code bloated in the new experiments, especially for Full Xover. To deal with this, we terminated a run when the size of each program in a population was significantly bigger than 31 (for instance, 127 nodes) and the situation lasted for ten generations. Finally we measured the performance in the new experiments and analysed the DCP and the SSS of GPPs whose sizes are no more than 31 nodes.

Table 6.6 shows the performance measures in the new experiments for the three problems respectively. For EvePar and SymReg, the performance trends in GP systems using different crossover modes are consistent with these in the experiments with the size-limiting. For BinCla, a slight difference exists between the two sets of experiments. The test error rate comparison result between the GP systems using Partial$^-$ and Partial$^+$ Xovers in the new experiments are opposite to that in the experiments with the size-limiting, but the differences between them are very small. Overall, the performance trends in two sets of experiments are consistent.

Table 6.6: Performance comparison.

Xover Mode	EvePar	SymReg	BinCla
	Failure	RMS Error	Test Error Rate (%)
Standard	48	38.3 ± 8.2	5.3 ± 1.6
Partial$^-$	34	21.7 ± 7.2	4.4 ± 1.5
Partial	4	6.1 ± 3.4	4.7 ± 1.4
Partial$^+$	0	1.3 ± 1.9	4.9 ± 1.5
Full	0	0.1 ± 0.1	4.5 ± 1.3

Figures 6.11 and 6.12 illustrate the distributions of absolute depths of crossover points and absolute subtree sizes for GPPs involved in crossover in the five GP systems for the three problems. GPPs whose sizes are larger than 31 nodes are filtered out. The frequency calculation and the rescale method are the same as those used in the previous DCP and SSS analyses for the experiments with the size-limiting.

By comparing these two figures with Figures 6.5 and 6.8 respectively, we noticed some variations between them. However, the variations are very small, especially in Full Xover, and mainly related to few outlier parent programs of the largest depth or size, giving limited impact on the overall consistency between the two sets of experiments.

In summary, the distributions of absolute depths of crossover points and absolute subtree sizes obtained in the two sets of experiments are consistent. The impact of the size-limiting method on the findings obtained is negligible.

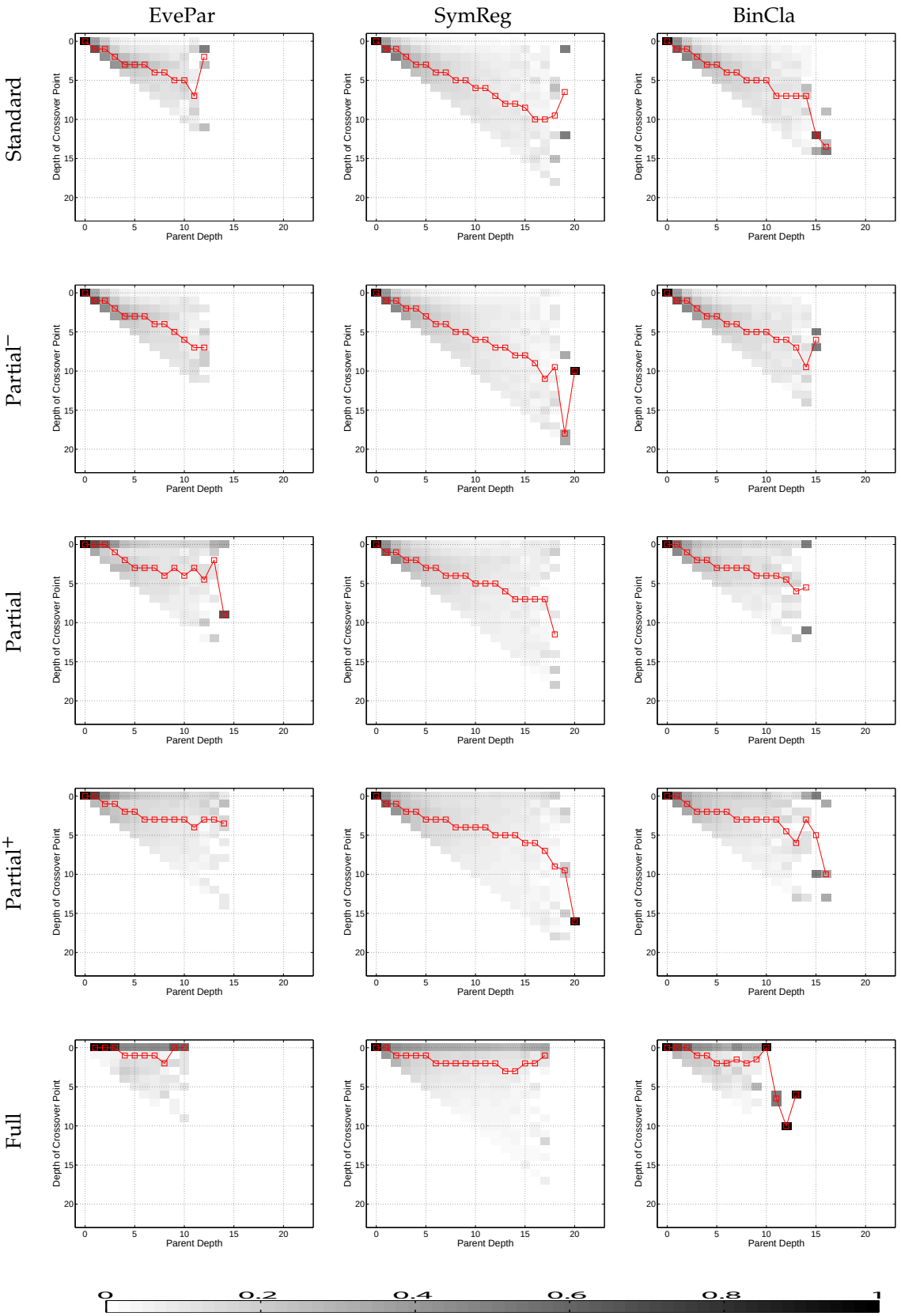


Figure 6.11: Distributions of absolute depths of crossover points for GPPs involved in crossover partitioned by parent depth and rescaled within each partition. Median values are highlighted by linked squares. GPPs of sizes larger than 31 nodes are filtered out.

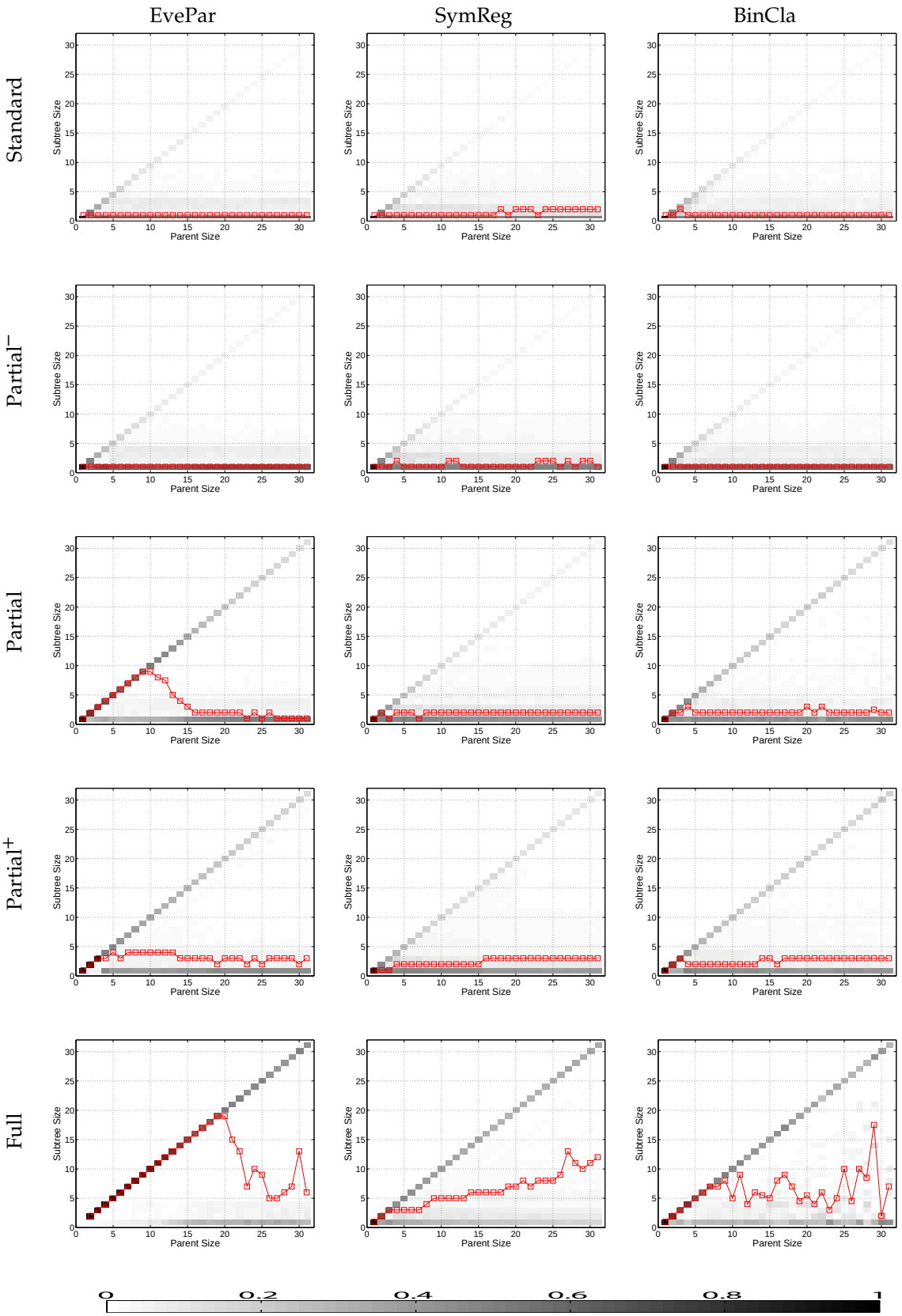


Figure 6.12: Distributions of absolute subtree sizes for GPPs involved in crossover partitioned by parent tree size and rescaled within each partition. Median values are highlighted by linked squares. GPPs of sizes larger than 31 nodes are filtered out.

6.10 Chapter Summary

This chapter conducted analyses of the depth of crossover point and the substituted subtree size in GPPs to investigate general heuristics for reducing the offspring search space. It presented four approximations of the optimal crossover operator and identified the most effective one based on the experiments on three problems in different domains. The DCP and SSS analysis results from the GP systems using the standard crossover and the four simulations of the optimal crossover operators, especially Full Xover, provided the following findings and heuristics:

- When the root of a program tree is allowed to be a crossover point, the analyses of the best approximation show that good crossover events have a strong preference for whole program trees, and (less strongly) single-node or small subtrees that are at the bottoms of parent program trees. Therefore, a good depth control strategy should have unequal rather than equal probabilities.

- The distributions of subtree size ratios are more consistent across different problems than the distributions of depth ratios, but are still evolutionary stage-dependent, as are the depth ratios. Taking into account both depth and subtree size in designing crossover point selection strategies should provide better performance in reducing offspring search space than should using either alone.

In addition, this chapter also provided some other interesting findings from the analysis:

- If a pair of given parents could not produce better offspring after a sufficient number of tries, a GP system should retain parents in the search space rather than select compromised crossover points that may lead the search nowhere.

- From the literature, the standard crossover operator has been shown to be destructive but there is no clear quantitative description of how destructive

it is. In our experiments, according to the analyses of Full Xover, the ability of standard GP crossover to generate good offspring has been quantified for given problems and is far below what was expected.

- Having the root as a potential crossover point and assigning it a high selection probability is beneficial because it enables an entire program tree to be used as a building block or a high-level new feature in other programs.

Note that all these findings are based on the experiments using the size-limiting bloat control method. Although this chapter has demonstrated that the impact of the size-limiting bloat control method appears to be very small, it would still be useful to test these findings using different bloat control methods as listed in [34].

Furthermore, as in the previous chapter, mutation operators were not used in all experiments. It would be necessary to verify the findings by including mutation operators, as well as disallowing roots to be potential crossover points, in the future.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Genetic programming, one of the metaheuristic search methods in evolutionary algorithms, is based on the Darwinian natural selection theory. "Survival of the fittest" has driven EAs since the 1950s and many selection mechanisms have been developed. However, how to select parent states and how to move within the immediate neighbourhood search space of given parent states remain important open issues. The overall goal of the thesis was to analyse selection behaviour for building an effective and efficient tree-based GP system.

This thesis has achieved its overall goal. It provided a detailed understanding of selection through analyses of the selection process in tree-based GP, covering both parent and offspring selection. It developed three novel methods and offered some guidance on improving GP search effectively and efficiently.

7.1.1 General Conclusions

The thesis showed that population size is not a factor when determining tournament size for an intended parent selection pressure. It clarified that the multi-sampled and the not-sampled issues are not critical in standard tournament selection. It showed that a promising way to tune parent selection pressure dynamically and automatically along evolution is to use the knowledge of FRD of a population, instead of using different sampling replacement strategies. It ob-

served that premature convergence occurs more often when stochastic elements are removed from both parent and offspring selection processes, and concluded that it is better to apply high offspring selection pressure while using low parent selection pressure.

The thesis also developed three novel methods. The clustering tournament selection method can tune parent selection pressure automatically and dynamically along evolution. The Ejit (evaluated-just-in-time) method can save fitness evaluation cost for standard tournament selection using small tournament sizes. It is a natural, simple and effective method requiring neither extra memory nor any pre- or post- processes like EMS-EA or BC-EA. The heuristic fitness-case-equivalence population clustering method can also save fitness evaluation cost and exceeds the limitations of Ejit. It can take advantage of the clustering tournament selection and is independent of tournament sizes.

Furthermore, the thesis indicated that a significant reduction in fitness evaluation cost in the parent selection phase could be achieved if correlates of GPPs (good predecessor programs) can be identified. It also observed that good crossover events have a strong preference for whole program trees, and (less strongly) single-node or small subtrees that are at the bottom of parent program trees. It suggested that considering both depth and subtree size should be able to provide better performance in reducing offspring search space than should the use of either alone.

7.1.2 Specific Conclusions

The five research questions given in Chapter 1 (on page 3) are answered below:

1. **How should parent selection pressure be properly controlled?**

   As tournament selection is the most commonly used parent selection scheme in GP, answering this question requires an understanding of the relationship between population size, tournament size, and selection pressure, as well as the multi-sampled, the not-sampled, and the high between-group selection pressure issues in standard tournament selection.

   The thesis conducted an extensive analysis via modelling and visualisation

to reveal the working of standard tournament selection. It showed that selection pressure is controlled mainly by tournament size and is aggravated by a skewed FRD (fitness rank distribution), but is independent of population size. As a result, it is not necessary to take into account population size when designing an intended parent selection pressure.

The thesis investigated the multi-sampled and the not-sampled issues in standard tournament selection through simulations and experiments and showed that these two issues are not critical to the selection behaviour in standard tournament selection. Different sampling replacement strategies have little impact on the parent selection pressure and cannot tune selection pressure in dynamic evolution.

The thesis analysed the high between-group selection pressure issue in standard tournament selection and showed that it is part of the general dynamic evolutionary learning process. Parent selection pressure needs to be controlled dynamically and automatically along evolution in order to improve the search performance. The thesis developed the clustering tournament selection method that takes knowledge of FRD at each generation and tunes parent selection pressure to meet the requirements of the dynamic evolutionary learning process.

2. **How can the cost of fitness evaluation in the parent selection process be minimised?**

   The thesis showed that the not-sampled characteristic of standard tournament selection (with low selection pressure) can be used to reduce the fitness evaluation cost in the parent selection phase through a novel, passive, and simple evaluation algorithm called Ejit. The thesis also showed that population can be clustered to reduce the fitness evaluation cost while taking advantage of the clustering tournament selection and avoiding the limitation of using small tournament sizes as in Ejit. Furthermore, the thesis conducted an initial analysis on the feasibility of using GPPs to minimise the fitness evaluation cost and indicated that the efficiency of parent selection could be improved as much as possible if correlates of GPPs can be

identified.

3. **How does applying offspring selection together with a parent selection mechanism affect GP search results?**

   The thesis conducted two sets of experiments to investigate the impact of offspring selection on the overall GP search performance in the context of using only crossover operators. Six different combinations of parent and offspring selection pressure configurations were tested. The thesis showed that applying selection pressure towards good offspring is better than no offspring selection pressure, but GP systems end up with premature convergence more often when applying high offspring selection pressure together with parent selection pressure.

4. **How should parent selection pressure and offspring selection pressure be configured in order to significantly improve the effectiveness of GP search?**

   Applying selection pressure means reducing stochastic elements in GP search. Based on the experimental results for answering the previous research question, the thesis showed that stochastic elements cannot be removed in both the parent selection process and the breeding process. Increasing offspring selection pressure must be coupled with a decreased parent selection pressure. The total amount of stochastic elements in GP search must be kept at a certain level. The thesis showed that it is preferable to take stochastic elements away from offspring selection instead of from parent selection. The thesis also showed that a good practice of configuring parent and offspring selection pressure is to have high offspring selection pressure and low parent selection pressure.

5. **How can the exploration of good offspring search space be constrained structurally in order to minimise the fitness evaluation cost in the offspring selection process?**

   The thesis focused on two aspects of program structure — the depth of crossover point and the substituted subtree size — to analyse good crossover

events which were extracted from several simulations of the optimal crossover operator. When the root of a program tree is allowed to be a potential crossover point, the thesis showed that good crossover events have a strong preference for the root, and (less strongly) single-node or small subtrees that are at the bottom of parent program trees. The thesis also showed that the distributions of subtree size ratios are more consistent across different problems than the distributions of depth ratios, but that both are evolutionary stage-dependent. Therefore, a good heuristic for efficiently searching good offspring is to combine both unequal-probability depth and small subtree size strategies to reduce crossover search space.

The thesis also obtained several other interesting findings from the analyses which are highlighted below:

- It is well-known that the standard crossover operator can be destructive in GP. The thesis demonstrated that even using the best approximation of the optimal crossover operator, in over 30% of crossover events (over 60% for some problems) involving GPPs, none of the possible offspring was better than the parents. These results showed that the ability of standard GP crossover to generate good offspring is far below what was expected and it is even more destructive than is generally recognised.

- According to the usual definition of crossover, the root of a program tree is excluded from potential crossover points. The thesis showed that having the root as a potential crossover point and assigning it a high selection probability is beneficial because it enables an entire program tree to be used as a building block or a high-level new feature in other programs.

- When stochastic elements are preserved in parent selection, the thesis showed that if a pair of given parents could not produce any better offspring after a sufficient number of tries, a GP system should retain the parent states in the search space instead of directing the search to worse states.

7.2 Future Work

This section highlights the most significant directions for future work.

7.2.1 Investigating heuristics for developing robust population clustering algorithms

Chapter 3 showed a promising method for optimising parent selection pressure dynamically and automatically. In order to use the method, the population must be clustered in advance. Some problems may require clustering populations based on exact program structure and content; some problems may require clustering populations based on fitness precisely; and some problems may require clustering populations based on fitness in a fuzzier manner. An interesting direction for future work is to investigate heuristics to guide and develop robust population clustering algorithms for different kinds of problems.

7.2.2 Investigating a way to determine GPPs

Chapter 4 analysed the feasibility of using GPPs to increase parent selection efficiency and showed that the number of GPPs is smaller than 4% of the total evaluated programs in some cases. The parent fitness evaluation cost could be reduced as much as possible without affecting the effectiveness of GP search if GPPs could be identified in advance. Although it is challenging to identify GPPs directly without evaluating populations, an attractive idea for future work is to find correlates of GPPs in order to identify them with minimal cost.

7.2.3 Investigating an appropriate offspring search intensity

Chapter 5 showed that a good practice for optimising GP search performance is to combine high offspring selection pressure with low parent selection pressure. It is fine to set parent selection pressure as low as random but it is not necessary to use the highest offspring search intensity to search every immediate neighbourhood state of given parents, especially in a resource-limited GP search. Chapter 5 showed that a good practice is to search in a subset of all possible immediate

neighbourhood states. Implementing this will require investigating appropriate levels of offspring search intensity in the context of a resource-limited GP process.

7.2.4 Investigating correlations between crossover point depth and substituted subtree size via tree shape analysis

For a binary tree, no matter whether it is a full tree or not, the number of leaf nodes is always around 50% of the total number of nodes. However, leaf nodes can appear at any depth if the tree is not a full balanced tree. For trees including unary functions and functions with more than two arguments, the situation becomes more complicated. Therefore, for improving the standard crossover, adjusting the depth of crossover point by assigning probabilities (equal or unequal) to each depth without considering the shape of a program tree would not be sensible, and may produce the same outcome as does the standard crossover.

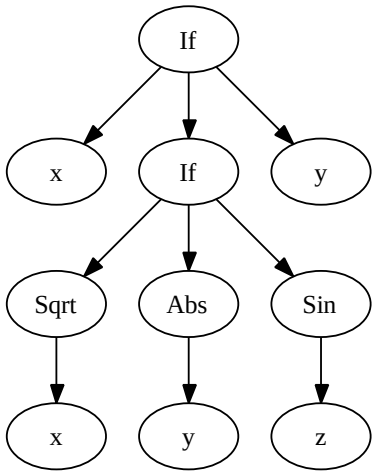


Figure 7.1: A sample program tree with same number of nodes at each depth (except the root).

For instance, for the parent program tree shown in Figure 7.1, if the root is not a possible crossover point, using the standard crossover, each node has the same probability of being selected as the crossover point and the probability of selecting a leaf node is 50%. If equal probability is assigned to each depth, then all nodes (except for the root) will still have the same probability of being selected, producing the same outcome as the standard crossover. If different probabilities are assigned to different depths, for instance giving even heavier weights to

higher nodes in a program tree, this should normally reduce the probability of selecting leaf nodes. However, in this shape tree, if giving 45% to depth one, 30% to depth two, and 25% to depth three, it even increases the probability of selecting leaf nodes (55%) compared with the standard crossover.

Therefore, future work needs to analyse tree shape to further investigate correlations between the depths and the substituted subtree sizes of preferred crossover points in order to clarify factors to guide crossover point selection.

7.2.5 Investigating impacts of mutation operators

Chapter 5 and Chapter 6 both focused only on crossover to investigate research questions. It might be possible to obtain different experimental results and findings if taking into consideration mutation operators. To verify the findings and conclusions presented in Chapters 5 and 6, future work should conduct further experiments using different mutation operators with various mutation rates.

Bibliography

[1] ABRAMOWITZ, M., AND STEGUN, I. A., Eds. *Handbook of Mathematical Functions*. Dover, New York, 1965.

[2] AGNELLI, D., BOLLINI, A., AND LOMBARDI, L. Image classification: an evolutionary approach. *Pattern Recognition Letters 23*, 1-3 (2002), 303–309.

[3] AKYOL, A., YASLAN, Y., AND EROL, O. K. A genetic programming classifier design approach for cell images. In *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU* (Hammamet, Tunisia, Oct. 31 - Nov. 2 2007), K. Mellouli, Ed., vol. 4724 of *Lecture Notes in Computer Science*, Springer, pp. 878–888.

[4] ALTENBERG, L. Emergent phenomena in genetic programming. In *Proceedings of the Third Annual Conference on Evolutionary Programming* (1994), A. V. Sebald and L. J. Fogel, Eds., World Scientific, pp. 233–241.

[5] ANDRE, D. The evolution of agents that build mental models and create simple plans using genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 July 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 248–255.

[6] ANDREAE, P., XIE, H., AND ZHANG, M. Genetic programming for detecting rhythmic stress in spoken english. *International Journal of Knowledge-Based and Intelligent Engineering Systems. Special Issue on Genetic Programming. 12*, 1 (2008), 15–28.

[7] ANDREWS, M., AND PRAGER, R. Genetic programming for the acquisition of double auction market strategies. In *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. MIT Press, 1994, ch. 16, pp. 355–368.

[8] ANGELINE, P. J. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 21–29.

[9] ANGELINE, P. J. Two self-adaptive crossover operators for genetic programming. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, Cambridge, MA, USA, 1996, ch. 5, pp. 89–110.

[10] ANGELINE, P. J. Subtree crossover: Building block engine or macromutation? In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (7 1997), pp. 9–17.

[11] ANGLUIN, D. Computational learning theory: survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing* (Victoria, British Columbia, Canada, 1992), ACM, pp. 351–369.

[12] BALAZS, M. E., AND RICHTER, D. L. A genetic algorithm with dynamic population: Experimental results. In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference* (Orlando, Florida, USA, 13 July 1999), S. Brave and A. S. Wu, Eds., pp. 25–30.

[13] BALDI, P., AND BRUNAK, S. *Bioinformatics: The Machine Learning Approach*, 2 ed. MIT Press, 2001.

[14] BANZHAF, W., NORDIN, P., KELLER, R., AND D. FRANCONE, F. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[15] BäCK, T. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation.* (1994), pp. 57–62.

[16] BEYER, H.-G. *The theory of Evolution Strategies*. Springer, 2001.

[17] BIRGE, L., AND ROZENHOLC, Y. How many bins should be put in a regular histogram. *European Series in Applied and Industrial Mathematics: Probability and Statistics 10* (2006), 24–45.

[18] BLICKLE, T., AND THIELE, L. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms* (1995), pp. 9–16.

[19] BLICKLE, T., AND THIELE, L. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation 4*, 4 (1997), 361–394.

[20] BOX, G., HUNTER, S., AND HUNTER, W. G. *Statistics for Experimenters: Design, Innovation, and Discovery*, 2nd ed. John Wiley, 2005.

[21] BRAMEIER, M., BANZHAF, W., AND INFORMATIK, F. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation 5* (2001), 17–26.

[22] BRANKE, J., ANDERSEN, H. C., AND SCHMECK, H. Global selection methods for SIMD computers. In *Proceedings of the AISB96 Workshop on Evolutionary Computing* (1996), pp. 6–17.

[23] BRINDLE, A. *Genetic algorithms for function optimisation*. PhD thesis, Department of Computing Science, University of Alberta, 1981.

[24] BULMER, M. *The Mathematical Theory of Quantitative Genetics*. Oxford University Press, Oxford, UK, 1980.

[25] BURKE, E., GUSTAFSON, S., AND KENDALL, G. A survey and analysis of diversity measures in genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference* (2002), pp. 716–723.

[26] CASTILLO, F., KORDON, A., SWEENEY, J., AND ZIRK, W. Using genetic programming in industrial statistical model building. In *Genetic Programming Theory and Practice II*, U.-M. O'Reilly and et al, Eds. Springer, 2006, ch. 3, pp. 31–48.

[27] CHAN, K. Y., AYDIN, M. E., AND FOGARTY, T. C. New factorial design theoretic crossover operator for parametrical problem. In *Genetic Programming, Proceedings of EuroGP'2003* (2003), C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610 of *LNCS*, Springer, pp. 22–33.

[28] CHELLAPILLA, K. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation 1*, 3 (Sept. 1997), 209–216.

[29] CHEN, Y., HU, J., HIRASAWA, K., AND YU, S. Gars: an improved genetic algorithm with reserve selection for global optimisation. In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), pp. 1173–1178.

[30] CHEN, Y., HU, J., HIRASAWA, K., AND YU, S. Solving deceptive problems using a genetic algorithm with reserve selection. In *Proceedings of IEEE Congress on Evolutionary Computation* (2008), pp. 884–889.

[31] CIESIELSKI, V., AND MAWHINNEY, D. Prevention of early convergence in genetic programming by replacement of similar programs. In *Proceedings of the 2002 Congress on Evolutionary Computation* (2002), IEEE Press, pp. 67–72.

[32] CORTES, C., AND VAPNIK, V. Support-vector network. *Machine learning 20* (1995), 273–297.

[33] COUCHET, J., MANRIQUE, D., RIOS, J., AND RODRIGUEZ-PATON, A. Crossover and mutation operators for grammar-guided genetic programming. *Soft Computing - A Fusion of Foundations, Methodologies and Applications 11*, 10 (2007), 943–955.

[34] DA SILVA, S. G. O. *Controlling Bloat: Individual and Population Based Approaches in Genetic Programming*. PhD thesis, University of Coimbra, 2008.

[35] DARK, G. On-line medical dictionary, 2005.

[36] DE JONG, K. Parameter setting in eas: a 30 year perspective. In *Parameter Setting in Evolutionary Algorithms*. Springer, 2007, pp. 1–18.

[37] DE SA, L. B., AND MESQUITA, A. Evolutionary synthesis of low-sensitivity equalizers using adjacency matrix representation. In *Proceedings of the 10th*

annual conference on Genetic and evolutionary computation (Atlanta, GA, USA, 12-16 July 2008), M. Keijzer, G. Antoniol, C. B. Congdon, K. Deb, B. Doerr, N. Hansen, J. H. Holmes, G. S. Hornby, D. Howard, J. Kennedy, S. Kumar, F. G. Lobo, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, and I. Wegener, Eds., ACM, pp. 1283–1290.

[38] DE SANTOS, P. G., GARCIA, E., AND ESTREMERA, J. *Quadrupedal Locomotion: An Introduction to the Control of Four-legged Robots*, 1 ed. Springer, June 2006.

[39] DEFOIN PLATEL, M., CLERGUE, M., AND COLLARD, P. Maximum homologous crossover for linear genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003* (2003), C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610 of *LNCS*, Springer, pp. 194–203.

[40] D'HAESELEER, P. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence* (Orlando, Florida, USA, 27-29 1994), vol. 1, IEEE Press, pp. 256–261.

[41] DORIGO, M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[42] EIBEN, A. E., AND SCHIPPERS, C. A. On evolutionary exploration and exploitation. *Fundamenta Informaticae 35*, 1-4 (1998), 35–50.

[43] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. Springer, 2003.

[44] EKÁRT, A., AND NÉMETH, S. Z. Maintaining the diversity of genetic programs. In *Proceedings of the 5th European Conference on Genetic Programming* (London, UK, 2002), Springer, pp. 162–171.

[45] ENACHE, R., SENDHOFF, B., OLHOFER, M., AND HASENJÄGER, M. Comparison of steady-state and generational evolution strategies for parallel architectures. In *Parallel Problem Solving from Nature – PPSN VIII* (2004), X. Yao et al., Eds., Lecture Notes in Computer Science 3242, Springer, pp. 253–262.

[46] FERNANDEZ, F., TOMASSINI, M., AND VANNESCHI, L. Saving computational effort in genetic programming by means of plagues. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation* (Canberra, 8-12 Dec. 2003), R. Sarker and et al, Eds., IEEE Press, pp. 2042–2049.

[47] FERREIRA, C. Gene expression programming: a new adaptive algorithm for solving problems. *COMPLEX SYSTEMS 13* (2001), 87.

[48] FILIPOVIĆ, V., KRATICA, J., TOŠIĆ, D., AND LJUBIĆ, I. Fine grained tournament selection for the simple plant location problem. In *5th Online World Conference on Soft Computing Methods in Industrial Applications* (2000), pp. 152–158.

[49] FISHER, D. H., PAZZANI, M., AND LANGLEY, P., Eds. *Concept Formation: Knowledge and Experience in Unsupervised Learning*. Morgan Kaufmann, 1991, ch. 1, p. 12.

[50] FOGEL, L. J., OWENS, A. J., AND WALSH, M. J. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[51] FRANCONE, F. D. Discipulus Owner's Manual. Freely available on the Web at the page `http://www.aimlearning.com/TechnologyOverview.htm`, 2000.

[52] FRANCONE, F. D. Discipulus owner's manual, 2004.

[53] GATHERCOLE, C. *An Investigation of Supervised Learning in Genetic Programming*. PhD thesis, University of Edinburgh, 1998.

[54] GATHERCOLE, C., AND ROSS, P. Dynamic training subset selection for supervised learning in genetic programming. In *In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, Parallel Problem Solving from Nature III* (1994), Springer-Verlag, pp. 312–321.

[55] GATHERCOLE, C., AND ROSS, P. An adverse interaction between crossover and restricted tree depth in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA,

USA, 28–31 July 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 291–296.

[56] GIACOBINI, M., TOMASSINI, M., AND VANNESCHI, L. Limiting the number of fitness cases in genetic programming using statistics. In *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature* (London, UK, 2002), Springer, pp. 371–380.

[57] GOLDBERG, D. E., AND DEB, K. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms* (1991), 69–93.

[58] GREFENSTETTE, J. J., AND BAKER, J. E. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (1989), J. D. Schaffer, Ed., Morgan Kaufmann Publishers, pp. 20–27.

[59] GRUAU, F. On using syntactic constraints with genetic programming. *Advances in genetic programming: volume 2* (1996), 377–394.

[60] GUSTAFSON, S. M. *An Analysis of Diversity in Genetic Programming*. PhD thesis, University of Nottingham, 2004.

[61] HADJAM, F. Z., MORAGA, C., AND BENMOHAMED, M. Cluster-based evolutionary design of digital circuits using improved multi-expression programming. In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), pp. 2475–2482.

[62] HANDLEY, S. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence* (Orlando, Florida, USA, 27-29 June 1994), vol. 1, IEEE Press, pp. 154–159.

[63] HARIK, G. R. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms* (San Francisco, CA, 1995), Morgan Kaufmann, pp. 24–31.

[64] HARRIES, K., AND SMITH, P. Exploring alternative operators and search strategies in genetic programming. In *Proceedings of the Second Annual Conference on Genetic Programming* (Stanford University, CA, USA, 13-16 1997), Morgan Kaufmann, pp. 147–155.

[65] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2 ed. Springer-Verlag, Dec. 2008.

[66] HAYNES, T. D., SCHOENEFELD, D. A., AND WAINWRIGHT, R. L. Type inheritance in strongly typed genetic programming. *Advances in Genetic Programming: volume 2* (1996), 359–376.

[67] HENERY, R. Classification. In *Machine Learning, Neural and Statistical Classification*, D. Michie, D. Spiegelhalter, and C. Taylor, Eds. Ellis Horwood, 1994, ch. 2, p. 6.

[68] HENGPROPROHM, S., AND CHONGSTITVATANA, P. Selective crossover in genetic programming. In *ISCIT International Symposium on Communications and Information Technologies* (ChiangMai Orchid, ChiangMai Thailand, 14-16 Nov. 2001).

[69] HINGEE, K., AND HUTTER, M. Equivalence of probabilistic tournament and polynomial ranking selection. In *Proceedings of IEEE Congress on Evolutionary Computation* (2008), pp. 564–571.

[70] HIRASAWA, K., OKUBO, M., KATAGIRI, H., HU, J., AND MURATA, J. Comparison between Genetic Network Programming (GNP) and Genetic Programming (GP). In *Proceedings of the 2001 Congress on Evolutionary Computation* (2001), vol. 2, pp. 1276–1282.

[71] HO, S.-Y., AND CHEN, J.-H. A ga-based systematic reasoning approach for solving travelling salesman problems using an orthogonal array-based crossover. In *Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region* (Beijing, 2000), pp. 659–663.

[72] HO, S.-Y., SHU, L.-S., AND CHEN, H.-M. Intelligent genetic algorithm with a new intelligent crossover using orthogonal arrays. In *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), vol. 1, pp. 289–296.

[73] HOLDENER, E. A., AND TAURITZ, D. R. Learning offspring optimising mate selection. In *Proceedings of Genetic and Evolutionary Computation Conference* (2008), pp. 1109–1110.

[74] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[75] HOLLAND, J. H. Adaptation. *Progress in Theoretical Biology 4* (1976).

[76] HONG, J.-H., AND CHO, S.-B. Lymphoma cancer classification using genetic programming with snr features. In *Proceedings of 7th EuroGP Conference* (2004), pp. 78–88.

[77] HOWARD, D., ROBERTS, S. C., AND BRANKIN, R. Target detection in SAR imagery by genetic programming. *Advances in Engineering Software 30* (1999), 303–311.

[78] HUTTER, M. Fitness uniform selection to preserve genetic diversity. Tech. Rep. IDSIA-01-01, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Manno(Lugano), CH, Jan. 2001.

[79] IBA, H., AND DE GARIS, H. Extending genetic programming with recombinative guidance. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, Cambridge, MA, USA, 1996, ch. 4, pp. 69–88.

[80] ISCID. Iscid encyclopaedia of science and philosophy, 2006.

[81] ITO, T., IBA, H., AND SATO, S. Depth-dependent crossover for genetic programming. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence* (Anchorage, Alaska, USA, 5-9 May 1998), IEEE Press, pp. 775–780.

[82] ITO, T., IBA, H., AND SATO, S. Non-destructive depth-dependent crossover for genetic programming. In *Proceedings of the First European Workshop on Genetic Programming* (Paris, 14-15 Apr. 1998), W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., vol. 1391 of *LNCS*, Springer, pp. 71–82.

[83] ITO, T., IBA, H., AND SATO, S. A self-tuning mechanism for depth-dependent crossover. In *Advances in Genetic Programming 3*, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, Eds. MIT Press, Cambridge, MA, USA, June 1999, ch. 16, pp. 377–399.

[84] JACKSON, D. Fitness evaluation avoidance in boolean GP problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Edinburgh, UK, 2-5 Sept. 2005), D. Corne and et al, Eds., vol. 3, IEEE Press, pp. 2530–2536.

[85] JAIN, A. K., MAO, J., AND MOHIUDDIN, K. M. Artificial neural networks: A tutorial. *IEEE Computer 29*, 3 (1996), 31–44.

[86] JIN, Y., AND SENDHOFF, B. Reducing fitness evaluations using clustering techniques and neural networks ensembles. In *Genetic and Evolutionary Computation Conference* (2004), vol. 3102 of *LNCS*, Springer, pp. 688–699.

[87] JONES, J., AND SOULE, T. Comparing genetic robustness in generational vs. steady state evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (2006), pp. 143–150.

[88] JUELS, A., AND WATTENBERG, M. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Tech. Rep. CSD-94-834, Department of Computer Science, University of California at Berkeley, USA, 18 1995.

[89] KENNEDY, J., AND EBERHART, R. Particle swarm optimisation. In *Proceedings of IEEE International Conference on Neural Networks* (1995), vol. 4, pp. 1942–1948.

[90] KENNEDY, J., AND EBERHART, R. C. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[91] KIM, H.-S., AND CHO, S.-B. An efficient genetic algorithms with less fitness evaluation by clustering. In *Proceedings of IEEE Congress on Evolutionary Computation* (2001), IEEE, pp. 887–894.

[92] KINNEAR, JR., K. E. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence* (Orlando, Florida, USA, 1994), vol. 1, IEEE Press, pp. 142–147.

[93] KOZA, J. R. *Genetic Programming — On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[94] KOZA, J. R. A response to the ML-95 paper entitled "Hill climbing beats genetic search on a boolean circuit synthesis of Koza's". Distributed 11 July 1995 at the 1995 International Machine Learning Conference in Tahoe City, California, USA, 11 July 1995.

[95] KOZA, J. R., III, F. H. B., ANDRE, D., AND KEANE, M. A. *Genetic Programming III: Darwinian Invention and Problem Solving*, 1st ed. Morgan Kaufmann, May 1999.

[96] KOZA, J. R., KEANE, M. A., STREETER, M. J., MYDLOWEC, W., YU, J., AND LANZA, G. *Genetic programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic, 2003.

[97] LANG, K. J. Hill climbing beats genetic search on a boolean circuit synthesis of Koza's. In *Proceedings of the Twelfth International Conference on Machine Learning* (Tahoe City, California, USA, July 1995), Morgan Kaufmann.

[98] LANGDON, W., AND POLI, R. Fitness causes bloat: Mutation. In *Proceedings of the First European Workshop on Genetic Programming* (Paris, 14-15 1998), W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., vol. 1391, Springer-Verlag, pp. 37–48.

[99] LANGDON, W. B. Size fair and homologous tree genetic programming crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), vol. 2, pp. 1092–1097.

[100] LANGDON, W. B., AND POLI, R. *Foundations of Genetic Programming*. Springer, Berlin, 2002.

[101] LANGDON, W. B., AND QURESHI, A. *Genetic programming - computers using natural selection' to generate programs*. University College London, Gower Street, London WC1E 6BT, UK, 1995. Research Note RN/95/76.

[102] LAW, N. L., AND SZETO, K. Adaptive genetic algorithm with mutation and crossover matrices. In *Proceedings of the International Joint Conference on Artificial Intelligence 2007* (2007), pp. 2330–2333.

[103] LEE, W.-C. Genetic programming decision tree for bankruptcy prediction. In *Proceedings of the 2006 Joint Conference on Information Sciences, JCIS 2006* (Kaohsiung, Taiwan, ROC, Oct. 8-11 2006), Atlantis Press.

[104] LEUNG, Y. W., AND WANG, Y. P. An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation 5*, 1 (February 2001), 41–53.

[105] LEVENE, M. *An Introduction to Search Engines and Web Navigation*. Pearson, 2005.

[106] LI, J., AND TSANG, E. P. K. Reducing failures in investment recommendations using genetic programming. In *Computing in Economics and Finance* (Universitat Pompeu Fabra, Barcelona, Spain, 6-8 July 2000).

[107] LINDEMAN, R. H., MERENDA, P. F., AND GOLD, R. Z. *Introduction to Bivariate and Multivariate Analysis*. Scott, Foresman and Company, 1980.

[108] LOBO, F. G., LIMA, C. F., AND MICHALEWICZ, Z., Eds. *Parameter Setting in Evolutionary Algorithms*, vol. 54 of *Studies in Computational Intelligence*. Springer, 2007.

[109] LOVEARD, T., AND CIESIELSKI, V. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation* (2001), vol. 2, IEEE Press, pp. 1070–1077.

[110] LUKE, S., BALAN, G. C., AND PANAIT, L. Population implosion in genetic programming. In *Proceedings of the 2003 conference on Genetic and Evolutionary Computation* (2003), vol. 2724 of *LNCS*, Springer, pp. 1729–1739.

[111] LUKE, S., AND PANAIT, L. Fighting bloat with nonparametric parsimony pressure. In *Proceedings of Parallel Problem Solving from Nature* (2002), vol. 2439 of *LNCS*, Springer, pp. 411–421.

[112] LUKE, S., AND PANAIT, L. Lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2002), pp. 829–836.

[113] LUKE, S., AND SPECTOR, L. A revised comparison of crossover and mutation in genetic programming. In *Proceedings of the 3rd annual conference on genetic programming* (1998), pp. 208–213.

[114] MAHFOUD, S. W. Crowding and preselection revisited. In *Parallel problem solving from nature 2* (Amsterdam, 1992), R. Männer and B. Manderick, Eds., North-Holland, pp. 27–36.

[115] MAJEED, H., AND RYAN, C. A less destructive, context-aware crossover operator for gp. In *Proceedings of EuroGP 2006* (2006), vol. 3905 of *LNCS*, Springer, pp. 36–48.

[116] MAKHOUL, J., STARNER, T., SCHWARTZ, R., AND CHOU, G. On-line cursive handwriting recognition using speech recognition methods. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing* (1994), vol. 5, pp. 125—128.

[117] MANNING, C. D., AND SCHUTZE, H. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[118] MANRIQUE, D., MARQUEZ, F., RIOS, J., AND RODRIGUEZ-PATON, A. Grammar based crossover operator in genetic programming. In *Proceedings of the 1st International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II* (2005), pp. 252–261.

[119] MATSUI, K. New selection method to improve the population diversity in genetic algorithms. In *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics* (1999), IEEE, pp. 625–630.

[120] MATTIUSSI, C., WAIBEL, M., AND FLOREANO, D. Measures of diversity for populations and distances between individuals with highly reorganizable genomes. *Evolutionary Computation 12*, 4 (2004), 495–515.

[121] MICHALSKI, R. S., CARBONELL, J. G., AND MITCHELL, T. M. *Machine Learning, An Artificial Intelligence Approach*. Tioga Publishing Company, California, 1983.

[122] MILLER, B. L., AND GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. Tech. Rep. 95006, University of Illinois at Urbana-Champaign, July 1995.

[123] MILLER, B. L., AND GOLDBERG, D. E. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation 4*, 2 (1996), 113–131.

[124] MILLER, J. F., JOB, D., AND THOMSON, P. Cartesian genetic programming. In *Proceedings of EuroGP 2000* (2000), vol. 1802 of *LNCS*, Springer, pp. 131–132.

[125] MITCHELL, M. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[126] MITCHELL, T. M. *Machine learning*. McGraw Hill, 1997.

[127] MOTOKI, T. Calculating the expected loss of diversity of selection schemes. *Evolutionary Computation 10*, 4 (2002), 397–422.

[128] MÜHLENBEIN, H., AND PAASS, G. From recombination of genes to the estimation of distributions i. binary parameters. In *Proceedings of the 4th Inter-*

national Conference on Parallel Problem Solving from Nature (1996), Springer-Verlag, pp. 178–187.

[129] MUHLENBEIN, H., AND SCHLIERKAMP-VOOSEN, D. Predictive models for the breeder genetic algorithm, I: continuous parameter optimization. *Evolutionary Computation 1*, 1 (1993), 25–49.

[130] MULLER, B., REINHARDT, J., AND STRICKLAND, M. T. *Neural Networks: An Introduction*, 2nd ed. Springer, Berlin Heidelberg, Germany, 1995.

[131] NEWMAN, D., HETTICH, S., BLAKE, C., AND MERZ, C. UCI repository of machine learning databases, 1998.

[132] NGUYEN, T. H., AND NGUYEN, X. H. A brief overview of population diversity measures in genetic programming. In *Proceedings of the Third Asian-Pacific workshop on Genetic Programming* (Military Technical Academy, Hanoi, VietNam, 2006), T. L. Pham, H. K. Le, and X. H. Nguyen, Eds., pp. 128–139.

[133] NORDIN, P., AND BANZHAF, W. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 310–317.

[134] NORDIN, P., FRANCONE, F., AND BANZHAF, W. Explicitly defined introns and destructive crossover in genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), J. P. Rosca, Ed., pp. 6–22.

[135] OLTEAN, M. Multi-expression programming. Tech. rep., Babes-Bolyai Univ, Romania, 2006.

[136] OLTEAN, M., AND GROSAN, C. Evolving evolutionary algorithms using multi expression programming. In *Proceedings of the 7th European Conference on Artificial Life* (2003), vol. 2801 of *LNAI*, Springer, pp. 651–658.

[137] O'NEILL, M., AND RYAN, C. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In *Genetic Programming*

7th European Conference, EuroGP 2004, Proceedings (Coimbra, Portugal, 5-7 Apr. 2004), M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., vol. 3003 of *LNCS*, Springer-Verlag, pp. 138–149.

[138] O'REILLY, U.-M., AND OPPACHER, F. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. *Lecture Notes in Computer Science 866* (1994), 397–406.

[139] O'REILLY, U.-M., AND OPPACHER, F. A comparative analysis of gp. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, 1996, ch. 2, pp. 23–44.

[140] PASSINO, K. M. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine 22*, 3 (2002), 52–67.

[141] PENA-REYES, C. A., AND SIPPER, M. Evolutionary computation in medicine: an overview. *Artificial Intelligence in Medicine 19*, 1 (2000), 1–23.

[142] POLADIAN, L. Excluding the best and worst individuals from parent selection. In *Proceedings of 2007 IEEE Congress on Evolutionary Computation* (2007), IEEE, pp. 400–406.

[143] POLI, R. Parallel distributed genetic programming. Tech. rep., School of Computer Science, University of Birmingham, 1996.

[144] POLI, R., AND LANGDON, W. B. Genetic programming with one-point crossover and point mutation. Tech. Rep. CSRP-97-13, University of Birmingham, UK, 1997.

[145] POLI, R., AND LANGDON, W. B. On the search properties of different crossover operators in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (22-25 July 1998), pp. 293–301.

[146] POLI, R., AND LANGDON, W. B. Backward-chaining evolutionary algorithms. *Artificial Intelligence 170*, 11 (2006), 953–982.

[147] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at

`http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[148] POLI, R., AND MCPHEE, N. F. Covariant parsimony pressure in genetic programming. Tech. Rep. CES-480, Department of Computing and Electronic System, University of Essex, 2008.

[149] POLI, R., MCPHEE, N. F., AND VANNESCHI, L. Elitism reduces bloat in genetic programming. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (2008), ACM Press, pp. 1343–1344.

[150] POPOVICI, E., AND JONG, K. D. Understanding EA dynamics via population fitness distributions. In *Proceedings of the Genetic and Evolutionary Computation Conference 2003* (2003), pp. 1604–1605.

[151] POPP, R. L., MONTANA, D. J., GASSNER, R. R., VIDAVER, G., AND IYER, S. Automated hardware design using genetic programming, VHDL, and FPGAs. In *IEEE International Conference on Systems, Man, and Cybernetics* (San Diego, CA USA, 11-14 Oct. 1998), vol. 3, IEEE, pp. 2184–2189.

[152] PRICE, K., STORN, R., AND LAMPINEN, J. *Differential Evolution - A Practical Approach to Global Optimization*. Springer, 2005.

[153] PRUGEL-BENNET, A., AND SHAPIRO, J. L. Analysis of genetic algorithms using statistical mechanics. *Phys. Rev. Lett. 72*, 9 (1994), 1305–1309.

[154] QUINLAN, J. *C4.5: Programs for machine learning*. Morgan Kaufmann, 1993.

[155] RECHENBERG, I. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, 1973.

[156] ROCHAT, D., TOMASSINI, M., AND VANNESCHI, L. Dynamic size populations in distributed genetic programming. In *Proceedings of the 8th European Conference on Genetic Programming* (Lausanne, Switzerland, 2005), vol. 3447 of *Lecture Notes in Computer Science*, Springer, pp. 50–61.

[157] ROSCA, J. P., AND BALLARD, D. H. Causality in genetic programming. In *Proceedings of the sixth international conference on genetic algorithms* (1995), Morgan Kaufmann, pp. 256–263.

[158] ROSCA, J. P., AND BALLARD, D. H. Rooted-tree schemata in genetic programming. *Advances in genetic programming: volume 3* (1999), 243–271.

[159] RUBINSTEIN, R., AND KROESE, D. *Simulation and the Monte Carlo Method*, second ed. John Wiley and Sons, 2007.

[160] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence — A modern approach*, 2nd ed. Pearson Education, New Jersey, US, 2003.

[161] SASTRY, K., GOLDBERG, D. E., AND PELIKAN, M. Don't evaluate, inherit. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, California, USA, 2001), L. Spector and et al, Eds., Morgan Kaufmann, pp. 551–558.

[162] SCHMIDT, M. D., AND LIPSON, H. Learning noise. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (London, 7-11 July 2007), D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, Eds., vol. 2, ACM Press, pp. 1680–1685.

[163] SCHOENAUER, M., LAMY, B., AND JOUVE, F. Identification of mechanical behaviour by genetic programming part II: Energy formulation. Tech. rep., Ecole Polytechnique, 91128 Palaiseau, France, 1995.

[164] SHERRAH, J. R., BOGNER, R. E., AND BOUZERDOUM, A. The evolutionary pre-processor: Automatic feature extraction for supervised classification using genetic programming. In *Proceedings of 2nd International Conference on Genetic Programming* (1997), pp. 304–312.

[165] SINGLETON, A. GPQUICK: A Simple Genetic Programming System in C++. http://www-cgi.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/gp/systems/gpquick/0.html.

[166] SMART, W., AND ZHANG, M. Applying online gradient descent search to genetic programming for object recognition. In *CRPIT '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation* (Dunedin, New Zealand, Jan. 2004), J. Hogan, P. Montague, M. Purvis, and C. Steketee, Eds., vol. 32 no. 7, Australian Computer Society, Inc., pp. 133–138.

[167] SMART, W., AND ZHANG, M. Applying online gradient descent search to genetic programming for object recognition. *Australian Computer Science Communications (Data Mining, CRPIT 32) 26* (January 2004), 133–138.

[168] SMART, W., AND ZHANG, M. Probability based genetic programming for multiclass object classification. In *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence* (2004), pp. 251–261.

[169] SMITH, P. W. H., AND HARRIES, K. Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation 6*, 4 (Winter 1998), 339–360.

[170] SMITH, S. F. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[171] SMITS, G., KORDON, A., VLADISLAVLEVA, K., JORDAAN, E., AND KOTANCHEK, M. Variable selection in industrial datasets using pareto genetic programming. In *Genetic Programming Theory and Practice III*, T. Yu, R. L. Riolo, and B. Worzel, Eds., vol. 9 of *Genetic Programming*. Springer, Ann Arbor, 12-14 May 2005, ch. 6, pp. 79–92.

[172] SMORODKINA, E., AND TAURITZ, D. Toward automating EA configuration: the parent selection stage. In *Proceedings of IEEE Congress on Evolutionary Computation* (2007), pp. 63–70.

[173] SOKOLOV, A., AND WHITLEY, D. Unbiased tournament selection. In *Proceedings of Genetic and Evolutionary Computation Conference* (2005), ACM Press, pp. 1131–1138.

[174] SONG, A., CIESIELSKI, V., AND WILLIAMS, H. Texture classifiers generated by genetic programming. In *Proceedings of the Congress on Evolutionary Computation* (2002), IEEE Press, pp. 243–248.

[175] SOULE, T., AND FOSTER, J. A. Code size and depth flows in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (Stanford University, CA, USA, 13-16 1997), J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds., Morgan Kaufmann, pp. 313–320.

[176] STORN, R., AND PRICE, K. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[177] TACKETT, W. A. Genetic programming for feature discovery and image discrimination. In *Proceedings of 5th International Conference on Genetic Algorithms* (1993), pp. 303–309.

[178] TACKETT, W. A. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 1994.

[179] TANESE, R. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (1989), J. D. Schaffer, Ed., Morgan Kaufmann Publishers, pp. 434–440.

[180] TELLER, A., AND VELOSO, M. A controlled experiment: evolution for learning difficult image classification. In *Proceedings of 7th Portuguese Conference on Artificial Intelligence* (1995), pp. 165–176.

[181] TERRIO, M., AND HEYWOOD, M. I. Directing crossover for reduction of bloat in GP. In *IEEE CCECE 2003: IEEE Canadian Conference on Electrical and Computer Engineering* (12-15 May 2002), W. Kinsner, A. Seback, and K. Ferens, Eds., IEEE Press, pp. 1111–1115.

[182] TERRIO, M. D., AND HEYWOOD, M. I. On naive crossover biases with reproduction for simple solutions to classification problems. In *Proceedings of Genetic and Evolutionary Computation Conference, Part II* (Seattle, WA, USA, 26-30 June 2004), K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, Eds., vol. 3103 of *Lecture Notes in Computer Science*, Springer, pp. 678–689.

[183] TETTAMANZI, A., AND TOMASSINI, M. *Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems*. Springer, Berlin Heidelberg New York, 2001.

[184] TETTAMANZI, A. G. B. Genetic programming without fitness. In *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996* (Stanford University, CA, USA, 28–31 July 1996), J. R. Koza, Ed., Stanford Bookstore, pp. 193–195.

[185] TOMASSINI, M., VANNESCHI, L., CUENDET, J., AND FERNANDEZ, F. A new technique for dynamic size populations in genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation* (2004), IEEE Press, pp. 486–493.

[186] TOPCHY, A., AND PUNCH, W. F. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (San Francisco, California, USA, 7-11 July 2001), L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., Morgan Kaufmann, pp. 155–162.

[187] VANNESCHI, L. *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Sciences, University of Lausanne, Switzerland, 2004.

[188] VANYI, R. Practical evaluation of efficient fitness functions for binary images. In *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC* (Lausanne,

Switzerland, 30 Mar.-1 Apr. 2005), F. Rothlauf and et al, Eds., vol. 3449 of *LNCS*, Springer, pp. 310–324.

[189] VAVAK, F., AND FOGARTY, T. C. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *In IEEE International Conference on Evolutionary Computation* (1996), pp. 192–195.

[190] VEKARIA, K. P. *Selective Crossover As an Adaptive Strategy for Genetic Algorithms*. PhD thesis, University College, London, 1999.

[191] WATSON, R., HORNBY, G. S., AND POLLACK, J. B. Modelling building-block interdependency. In *Proceedings of PPSN V* (1998), pp. 97–106.

[192] WEINBRENNER, T. GPC++ - Genetic Programming C++ Class Library, 1997.

[193] WHIGHAM, P. A. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (Tahoe City, California, USA, 9 July 1995), J. P. Rosca, Ed., pp. 33–41.

[194] WHITELY, D. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (1989), J. D. Schaffer, Ed., Morgan Kaufmann Publishers, pp. 116–121.

[195] WINKELER, J. F., AND MANJUNATH, B. S. Genetic programming for object detection. In *Proceedings of 2nd International Conference on Genetic Programming* (1997), pp. 330–335.

[196] WINKLER, R. L. *Introduction to Bayesian Inference and Decision*, 2nd ed. Probabilistic, 2003.

[197] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Diego, CA, 2000.

[198] WONG, M. L. Evolving recursive programs by using adaptive grammar based genetic programming. *Genetic Programming and Evolvable Machines 6*, 4 (Dec. 2005), 421–455.

[199] WONG, P., AND ZHANG, M. Scheme: Caching subtrees in genetic programming. In *Proceedings of IEEE Congress on Evolutionary Computation* (2008), pp. 2683–2690.

[200] XIE, H., ZHANG, M., AND ANDREAE, P. Automatic selection pressure control in genetic programming. In *Proceedings of the sixth International conference on Intelligent Systems Design and Applications* (2006), IEEE Computer Society Press, pp. 435–440.

[201] XIE, H., ZHANG, M., AND ANDREAE, P. Population clustering in genetic programming. In *Proceedings of the 9th European Conference, EuroGP 2006* (2006), vol. 3905 of *LNCS*, Springer, pp. 190–201.

[202] XIE, H., ZHANG, M., AND ANDREAE, P. An analysis of constructive crossover and selection pressure in genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), pp. 1739–1746.

[203] XIE, H., ZHANG, M., AND ANDREAE, P. An analysis of depth of crossover points in tree-based genetic programming. In *Proceedings of IEEE Congress on Evolutionary Computation* (2007), pp. 4561–4568.

[204] XIE, H., ZHANG, M., AND ANDREAE, P. Another investigation on tournament selection: modelling and visualisation. In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), pp. 1468–1475.

[205] XIE, H., ZHANG, M., AND ANDREAE, P. An analysis of the distribution of swapped subtree sizes in tree-based genetic programming. In *Proceedings of IEEE Congress on Evolutionary Computation* (2008), IEEE Press, pp. 2864–2971.

[206] XIE, H., ZHANG, M., ANDREAE, P., AND JOHNSTON, M. An analysis of multi-sampled issue and no-replacement tournament selection. In *Proceed-*

ings of Genetic and Evolutionary Computation Conference (2008), ACM Press, pp. 1323–1330.

[207] XIE, H., ZHANG, M., ANDREAE, P., AND JOHNSTON, M. Is the not-sampled issue in tournament selection critical? In *Proceedings of IEEE Congress on Evolutionary Computation* (2008), IEEE Press, pp. 3711–3718.

[208] YUEN, C. C. Selective crossover using gene dominance as an adaptive strategy for genetic programming. Msc intelligent systems, University College, London, UK, Sept. 2004.

[209] ZHANG, B. T., AND MUHLENBEIN, H. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex System 7* (1993), 199–220.

[210] ZHANG, B. T., AND MUHLENBEIN, H. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation 3*, 1 (1995), 17–38.

[211] ZHANG, B. T., OHM, P., AND MUHLENBEIN, H. Evolutionary induction of sparse neural trees. *Evolutionary Computation 5*, 2 (1997), 213–236.

[212] ZHANG, M., CIESIELSKI, V., AND ANDREAE, P. A domain independent window-approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing 2003*, 8 (2003), 841–859.

[213] ZHANG, M., GAO, X., AND LOU, W. Gp for object classification: Brood size in brood recombination crossover. In *The 19th Australian Joint Conference on Artificial Intelligence* (2006), vol. 4303 of *LNAI*, Springer, pp. 274–284.

[214] ZHANG, M., GAO, X., AND LOU, W. A new crossover operator in genetic programming for object classification. *IEEE Transactions on Systems, Man and Cybernetics, Part B 37*, 5 (Oct. 2007), 1332–1343.

[215] ZHANG, M., AND SMART, W. Multiclass object classification using genetic programming. In *Applications of Evolutionary Computing, EvoWorkshops2004* (2004), vol. 3005 of *LNCS*, Springer, pp. 369–378.

[216] ZHANG, Q., AND LEUNG, Y. W. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Transactions on Evolutionary Computation 3*, 1 (April 1999), 53–62.

[217] ZHANG, W., MING WU, Z., AND KE YANG, G. Genetic programming-based chaotic time series modeling. *Journal of Zhejiang University Science 5*, 11 (2004), 1432–1439.

[218] ZIEGLER, J., AND BANZHAF, W. Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function. In *Genetic Programming, Proceedings of EuroGP'2003* (2003), C. Ryan and et al, Eds., vol. 2610 of *LNCS*, Springer, pp. 264–275.

Appendix A

Proof of Equations 3.16 and 3.21 Being Equivalent

Proof. Equations 3.21 can be simplified to:

$$
\begin{aligned}
P(W_j) &= \frac{\sum_{n=1}^{k} \frac{1}{n} \frac{(|S_j|-1)!}{(n-1)!(|S_j|-1-n+1)!} \begin{pmatrix} \sum_{i=1}^{j-1} |S_i| \\ k-n \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix}} \\
&= \frac{\sum_{n=1}^{k} \frac{(|S_j|-1)!}{n!(|S_j|-n)!} \begin{pmatrix} \sum_{i=1}^{j-1} |S_i| \\ k-n \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix}} \\
&= \frac{\sum_{n=1}^{k} \frac{1}{|S_j|} \frac{|S_j|!}{n!(|S_j|-n)!} \begin{pmatrix} \sum_{i=1}^{j-1} |S_i| \\ k-n \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix}} \\
&= \frac{\sum_{n=1}^{k} \begin{pmatrix} |S_j| \\ n \end{pmatrix} \begin{pmatrix} \sum_{i=1}^{j-1} |S_i| \\ k-n \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix} |S_j|}
\end{aligned}
$$

After applying the relation $\sum_{b=0}^{a} \binom{x}{b} \binom{y}{a-b} = \binom{x+y}{a}$ [1] (page 822), we can further simplify the equation to

$$= \frac{\binom{|S_j| + \sum_{i=1}^{j-1} |S_i|}{k} - \binom{|S_j|}{0} \binom{\sum_{i=1}^{j-1} |S_i|}{k}}{\binom{N}{k} |S_j|}$$

$$= \frac{\binom{\sum_{i=1}^{j} |S_i|}{k} - \binom{\sum_{i=1}^{j-1} |S_i|}{k}}{\binom{N}{k} |S_j|}$$

which is the same as Equation 3.16. □

Appendix B

Glossary of Terms

BC-EA	Backward-Chaining Evolutionary Algorithm
DCP	Depth of Crossover Point
Ejit	Evaluated-just-in-time, a simple algorithm for saving the fitness evaluation cost in standard tournament selection using small tournament sizes
EMS-EA	Efficient Macro-Selection Evolutionary Algorithm
FCGP	a GP system in which the population is clustered based on Fitness and the parent selection is the Clustering tournament selection
FRD	Fitness Rank Distribution
GCGP	a GP system with the Genotype population clustering and the Clustering tournament selection
GPP	Good Predecessor Programs
HCC	Headless Chicken Crossover
HCGP	a GP system in which the population is clustered using the Heuristic fitness-case-equivalence population clustering algorithm and the parent selection method is the Clustering tournament selection
SSS	Substituted Subtree Size
Xover	Crossover