An Analysis of Diversity in Genetic Programming

by Steven Matt Gustafson

Thesis submitted to The University of Nottingham

for the degree of Doctor of Philosophy, February 2004

Contents

List of Figures

List of Tables

Abstract

Genetic programming is a metaheuristic search method that uses a population of variable-length computer programs and a search strategy based on biological evolution. The idea of automatic programming has long been a goal of artificial intelligence, and genetic programming presents an intuitive method for automatically *evolving* programs. However, this method is not without some potential drawbacks. Search using procedural representations can be complex and inefficient. In addition, variable sized solutions can become unnecessarily large and difficult to interpret.

The goal of this thesis is to understand the dynamics of genetic programming that encourages efficient and effective search. Toward this goal, the research focuses on an important property of genetic programming search: the population. The population is related to many key aspects of the genetic programming algorithm. In this programme of research, diversity is used to describe and analyse populations and their effect on search. A series of empirical investigations are carried out to better understand the genetic programming algorithm.

The research begins by studying the relationship between diversity and search. The effect of increased population diversity and a metaphor of search are then examined. This is followed by an investigation into the phenomenon of increased solution size and problem difficulty. The research concludes by examining the role of diverse individuals, particularly the ability of diverse individuals to affect the search process and ways of improving the genetic programming algorithm.

This thesis makes the following contributions: (1) An analysis shows the complexity of the issues of diversity and the relationship between diversity and fitness, (2) The genetic programming search process is characterised by using the concept of genetic lineages and the sampling of structures and behaviours, (3) A causal model of the varied rates of solution size increase is presented, (4) A new, tunable problem demonstrates the contribution of different population members during search, and (5) An island model is proposed to improve the search by speciating dissimilar individuals into better-suited environments.

Currently, genetic programming is applied to a wide range of problems under many varied contexts. From artificial intelligence to operations research, the results presented in this thesis will benefit population-based search methods, methods based on the concepts of evolution and search methods using variable-length representations.

Publications Produced

While pursing this Ph.D. research programme, I produced several publications that represented my ongoing research for the degree. The following publications are based on my research conducted for this thesis:

S. Gustafson, E.K. Burke, G. Kendall and N. Krasnogor. (in preparation). Diversity and Survivability in Genetic Programming. *IEEE Transactions on Evolutionary Computation.*

S. Gustafson, A. Ekart, E. Burke and G. Kendall. (to appear in 2004). Problem Difficulty and Code Growth in Genetic Programming. *Genetic Programming and Evolvable Machines.*

S. Gustafson, E.K. Burke and G. Kendall. (2004). Sampling of Unique Structures and Behaviours in Genetic Programming. In *Proceedings of the European Conference on Genetic Programming*. 10 pages, April, Coimbra, Spain. Springer.

E.K. Burke, **S. Gustafson**, G. Kendall. (2004). Diversity in Genetic Programming: An Analysis of Measures and Correlation with Fitness. *IEEE Transactions on Evolutionary Computation.* 8(1), pp. 47-62, IEEE Press.

E.K. Burke, **S. Gustafson**, G. Kendall and N. Krasnogor. (2003). Is Increasing Diversity in Genetic Programming Beneficial? An Analysis of the Effects on Fitness. In *Proceedings of the Congress on Evolutionary Computation*, pages 1398-1405, Canberra, Australia. IEEE Press.

E. Burke, **S. Gustafson**, G. Kendall and N. Krasnogor. (2002). Advanced Population Diversity Measures in Genetic Programming. In *Parallel Problem Solving from Nature.* Volume 2439 of LNCS, pages 341-350, Granada, Spain. Springer.

E. Burke, **S. Gustafson** and G. Kendall. (2002). A Survey and Analysis of Diversity Measures in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716-723, New York. Morgan Kaufmann Publishers.

Acknowledgments

I would like to thank my academic supervisors, Edmund Burke and Graham Kendall, for giving me the opportunity to study at the University of Nottingham. I am grateful for the many opportunities and guidance they have afforded me while at Nottingham. In particular, my research greatly benefited from the chance to attend several academic conferences.

Several people in the evolutionary algorithm and genetic programming community have provided invaluable discussions. I would like to especially thank Bill Hart, Bill Langdon, Sean Luke, Maarten Keijzer, Nic McPhee, Una-May O'Reilly, and Lee Spector.

My research benefited from collaborations with Atif Azad, Anikó Ekárt, Natalio Krasnogor and Leonardo Vanneschi.

In particular, I would like to thank Natalio Krasnogor for always having an open door to discuss, listen and help with my research. I also greatly appreciate the time my Dad gave in reviewing many drafts of papers and this thesis.

I also would like to acknowledge the support of my family in the United States, especially my parents, David and Karen.

Moving abroad and setting up home in a new country would not have been as enjoyable or bearable if it was not for my wife, Kristin. I thank her for her constant support and understanding.

To Kristin

and my Parents

CHAPTER 1

AN ANALYSIS OF DIVERSITY

This thesis examines the issue of diversity in genetic programming. Genetic programming, an evolutionary methodology in the fields of machine learning and artificial intelligence, uses a population of solutions to guide search. As such, population diversity is an important issue. This thesis describes a series of carefully planned and designed experiments that provide a detailed understanding of genetic programming, diversity and diversity related issues. An introduction to genetic programming is presented next, followed by the contributions and an overview of this thesis.

1.1 Introduction

Genetic programming uses a population of variable-length computer programs within the metaphor of evolution to guide *search* (Cramer, 1985; Koza, 1992). Search is a framework for problem solving where alternative solutions are evaluated in a trial-and-error fashion. A goal of search is to find an ordering of alternative solutions that leads to good solutions. When enumeration over the entire space of alternative solutions is not possible, heuristics like genetic programming use rules-of-thumb to guide the search for alternative solutions. Genetic programming belongs to a class of algorithms which use the metaphor of Natural Selection (Darwin, 1859) to determine the ordering of alternative solutions during search.

Genetic programming is a simple and powerful technique which has been applied to a wide range of problems in combinatorial optimisation, automatic programming and model induction (see (Banzhaf et al., 1998) for a general introduction to genetic programming). The direct encoding of solutions as variable-length computer programs allows genetic programming to provide solutions that can be evaluated and also examined to understand their internal workings. In this way, a genetic programming solution can represent a single value after evaluation (as in optimisation),

an iteratively built algorithm (as in automatic programming) or a data-driven model useful in data mining and knowledge discovery (as in model induction). While genetic programming is applied successfully in all three cases, each of which merits its own scientific discipline, the focus of this thesis is on the inner workings of the algorithm itself.

The issues of problem solving that motivate the use of genetic programming also represent some of the limits to its applicability. The increase in computer power, the wide range of applications and the ability to collect enormous amounts of scientific data all require a computational framework, such as genetic programming, capable of automatically generating solutions with a representation that is both powerful and understandable. However, the genetic programming algorithm has a high computational cost to run, has difficulty scaling to larger and harder problem instances, and uses a complex representation that can also limit its use.

At the heart of these issues is the use of a population of alternative solutions and the methods used to generate new alternatives. A candidate, or alternative, solution is evaluated by means of a *fitness* function that allows it to be compared with previously evaluated solutions. A stochastic selection method chooses better solutions from the population that then undergo stochastic variations to produce new alternatives. In this way, the population is responsible for guiding a parallel search through the solution space. However, the evaluation of each population member becomes increasingly computationally expensive and designing an effective process of variation on the direct and complex encoding of solutions is not intuitive. Also, the amount of data produced by a population-based search over an enormous space of possible programs makes the prediction and analysis of algorithm behaviour difficult.

1.2 Research Perspective

The focus of this thesis is on understanding the genetic programming population, the ways it can be measured and the role it plays on guiding the search process. Specifically, the diversity of the population is analysed to uncover key features and relationships that make search successful. In the process of developing a better understanding of genetic programming, a clearer description of the dynamics of the algorithm emerges to facilitate and motivate future enhancements. The existing theoretical models of evolutionary algorithms are limited in use and applicability due to their complexity. Therefore, the majority of theoretical work has been derived from experimentation. The approach taken in this thesis is also based on the careful design, collection and analysis of experimental results.

1.3 Contributions

This thesis makes the following contributions:

1. A survey and analysis of diversity in genetic programming demonstrates the complexity behind the issues of diversity measures and methods and the relationship between diversity and fitness.
2. An analysis using genetic lineages shows how a search metaphor of hill-climbing can be used to explain and improve genetic programming search. Also, the sampling of unique structures and behaviours by genetic programming demonstrates the low sampling of both complex behaviours and unique structures of large size.
3. A causal model is developed which links increased rates of code growth to non-decreased selection pressure and to increased similarity within the population. Decreased selection pressure occurs when fitness-based diversity is lost, and increased similarity in the population is the result of both faster convergence and non-decreased selection pressure.
4. An analysis using the Tree-String problem shows the inability to produce good offspring by both dissimilar-and-fit individuals and by similar-and-equally-well-fit individuals.
5. A model is proposed that identifies dissimilar individuals and moves them to new islands where they can contribute to search more effectively.

1.4 Overview

Chapter 2 introduces search, heuristics, evolutionary algorithms and genetic programming. An in-depth examination of genetic programming follows. The algorithm, representation and operators are described before an introduction is given to common benchmark problem domains. Two important research issues and a metaphor of genetic programming search are then discussed.

Chapter 3 discusses three major issues which are evaluated in this thesis. First, the representation and possible definitions of diversity are examined to highlight the complexity of diversity measures and methods. Secondly, the effects of diversity on other aspects of the search process is emphasised to demonstrate the wide-reaching issue of population diversity. The chapter concludes by discussing the role the population plays in producing new solutions to guide search is discussed.

Chapter 4 begins the first of four chapters which represent the original contributions in this thesis. Measures and methods of diversity are surveyed from the literature. Common measures are

used, along with novel population diversity measures, in an experimental study to assess the correlation between diversity and fitness at different stages of evolution. The results emphasise the contrasting behaviour between the common measures of diversity. Also, the most important diversity measures appear to be the ones that capture information relevant to other processes, such as selection and recombination.

Chapter 5 develops a diversity method based on genetic lineages. This method, lineage selection, is used to demonstrate how and why increasing the diversity of populations may be beneficial for some problems and not for others. Results suggest that many problems benefit or suffer from a hill-climbing type search. When hill-climbing is interrupted by lineage selection, performance is sometimes improved, sometimes worsened. To further investigate the type of search genetic programming performs, the sampling of unique behaviours and structures is analysed. Results provide a clearer characterisation of search on several problem domains.

Chapter 6 explores the effects diversity has on other aspects of the search process. A causal relationship is found between difficulty, diversity and code growth. Initially these results are reinforced by examining related literature, and then verified with a simplified model of genetic programming. The results indicate that increased difficulty leads to both non-decreased selection pressure and less structurally diverse populations, both of which contribute to an increased rate of code growth.

Chapter 7 explores the role of dissimilar individuals in the population and the effectiveness of migrant individuals in distributed models. Several definitions of genetic difference are used to probe these relationships on a new, constructed problem. The study looks at the ability of different subpopulations to produce offspring with high survivability. A survey of previous methods used to encourage distributed evolution is presented. A model is then proposed that explicitly identifies dissimilar solutions and places them into an environment where they will be most effective.

Chapter 8 states the conclusions, lists the contributions and summarises the problem-specific results obtained throughout the thesis. Recommendations for future research then follow.

CHAPTER 2

SEARCH, EVOLUTIONARY ALGORITHMS AND GENETIC PROGRAMMING

Evolutionary algorithms are heuristic search techniques within the broadly defined domain of artificial intelligence. Problem solving is a common application of artificial intelligence, where *search* is the framework for problem solving with computers. How does an evolutionary algorithm implement and carry out search? This chapter describes the process of search, the elements required to carry out search and different search strategies. Traditional search is described first, followed by common heuristic methods. The evolutionary algorithm is then described, showing how it addresses the task of problem solving by means of search. After introducing and describing genetic programming, the chapter concludes by discussing two important research issues: scalability and representation.

2.1 Problem Solving and Search

Most human activity can be defined as a form of complex problem solving. Thus, problem solving by a computer program is considered to be an instance of artificial intelligence, where the task consists of managing information and searching for solutions (Shapiro, 1990). Given a representation of a problem and a description of an ideal solution, the goal of search is to find a solution equal to or *close* to the ideal solution. In many cases, decision making and problem solving are formulated as optimisation problems (Pardalos and Resende, 2002). The role of the search method is to find the best solution among possible alternatives, optimising for solution quality.

An example of problem solving in artificial intelligence is the field of *automatic programming* (Barr et al., 1989). Computers perform calculations by means of programs. Designing and writing programs can be tedious and difficult. Given a specification of the desired behaviour of a program, it

would be desirable if another program could automatically generate a solution that met the specification. Although not normally considered an artifact of artificial intelligence, compilers were an early form of automatic programming as they allowed programmers to write an algorithm in a high-level language that was then interpreted and compiled into the lowel-level machine code used by the computer's processor. Later, while investigating the possibility of *learning machines*, Friedberg (1958) designed an algorithm that was able to evaluate the quality of a computer program, make random changes to it and then re-test for improvements. Since then, many advances have been made in the field of artificial intelligence and automatic programming. For a further discussion of automatic programming see (Barr et al., 1989).

Another example of problem solving by computer programs can be seen in the field of operational research. Real-world problems are routinely solved efficiently and accurately by means of computer programs (Pardalos and Resende, 2002). Operational research is concerned with studying such methods and their applications. The methods developed in the cross-disciplinary research between operational research and artificial intelligence are typically considered to be the state-of-the-art *applied* algorithms, as they produce the solutions that are used in everyday decision making.

2.1.1 Requirements of Search

To carry out search by means of a computer program, several elements of the problem and search strategy must be defined. Search requires the following elements (Nilsson, 1971):

- A defined problem, often requiring intelligent or complex behaviour to solve,
- A model of the problem and the representation of possible solutions,
- A goal state that defines the ideal solution, where heuristic methods often use an evaluation function to rank the non-goal state solutions,
- Transformation operators that are capable of changing an existing solution into an alternative solution, and
- A strategy for searching the space of possible solutions using the representation and transformation operators.

The application of the transformation operator(s) on a solution creates a *neighbourhood* of solutions. These new solutions can then be compared to the goal state. When knowledge of the problem is available, heuristic algorithms can define an evaluation function that allows the scoring and ranking of solutions. The ranking of solutions determines which, if any, of the solutions in the neighbourhood are better.

While search requires the above items, much of artificial intelligence and search research focuses on the last item, finding good search strategies.

2.1.2 Algorithms to Perform Search

With the ability to generate and evaluate a possible solution, a *random search* strategy can be defined. A random search method repeatedly generates solutions, evaluates them and generates more. Random search uses no knowledge of previously generated solutions and typically stops when a solution meets a particular criterion, such as being equal to the goal state or having quality above a threshold.

Transformation operators allow a solution to be transformed into one or many new solutions, depending on the number and type of operators. In this way, the neighbourhood of the current solution is generated. The neighbourhood represents all the solutions that can be generated by the application of the operators on the current solution.

Search strategies control the operators used, the size of the neighbourhood searched and which solutions are selected to continue the search from. In traditional search, where typically a goal state is defined, the operators can define a search tree. A node in the tree represents a solution, and its successors, or children, are defined by the operators. Strategies can then be defined to search the tree. Two such *blind* search strategies are *depth-first* and *breadth-first* search.

Breadth-first search generates all the successors of the root node in the search tree. Next, it generates all the successors of those nodes and continues until a solution is found. Depth-first search generates a single successor of the root node. Next, it generates one successor to that node and continues until a maximum depth is reached. At this point, it backtracks to the previous root node and, if possible, generates another successor.

For problems with large search trees, blind search methods become inefficient and impractical. This is particularly true when a search problem is cast as a decision problem (e.g. whether or not a solution exists with quality above a threshold) that is undecidable in polynomial time of the size of the problem instance. That is, if the decision problem is in the NP complexity class, the optimisation or search problem must also be at least as hard (Papadimitriou and Steiglitz, 1982).

However, blind search methods typically do not use any knowledge of the problem domain, which is often readily available. As it is common for problems to become intractably large, approximate methods called *heuristics* are typically used. Heuristics define rules-of-thumb about the problem structure, such as a way to generate a feasible solution in a combinatorial optimisation problem or a search strategy that finds good solutions in a reasonable amount of time (Reeves, 1995). Heuristics

can improve search efficiency by only considering a subset of all possible solutions or by only generating solutions that are likely to be better than the current solution.

A simple heuristic search method is *hill-climbing*. Hill-climbing (or steepest descent) begins with a randomly generated solution and continues to produce neighbours until a better solution is found. This new solution becomes the current solution and the algorithm begins producing its neighbours. If there are no improving solutions in the current neighbourhood, the method will stop. The current solution may or may not be a global optimum in this case. *Stochastic hill-climbing* is a variant of hill-climbing that proceeds in the same way but can also accept neighbours that are equivalent to the current solution. The *multi-start hill-climbing* method attempts to improve performance by performing several *trials* or *runs* of hill-climbing that each start from a different random initial solution. The operator used in heuristic techniques typically results in a small random change to the current solution. This type of operator, often called *mutation*, requires little or no domain knowledge. If all the neighbours of the current solution are created before selecting the best, the method is called *neighbourhood* search. *Variable neighbourhood* search adaptively increases the size of the neighbourhood when no improving solution is found.

Another simple heuristic search method is *best-first*. This method generates all the neighbours of a solution, selects the best one, generates all its neighbors, etc. When no better solution is found in a neighbourhood, the search returns to the previous neighbourhood and selects the second best, and continues until no improvements can be found. *Beam* search is similar to best-first, but does not remember the complete previous neighbourhood. Instead, beam search remembers a fraction of the best solutions in all previous neighbourhoods to return to when no improvements are found. These methods are also called *local search* methods as they only accept improving solutions that are in the *local* neighbourhood of the current solution.

By using knowledge of the domain, *heuristic evaluation functions* can be defined to provide an estimation of the quality of a current solution or the likelihood that the search will continue toward a goal state. Heuristic evaluation functions are also called cost functions, objective functions or fitness functions. The estimated solution quality reported by these functions is often called the *fitness* of the solution. With this measure of solution quality, the search strategy has additional information with which to guide search. *Metaheuristic methods* represent search strategies that use heuristic evaluation functions to guide search. Metaheuristics are iterative improvement algorithms that repeatedly transform solutions, selecting better *or* worse new solutions in order to improve the overall fitness of the final solution obtained. Metaheuristic methods assume that the heuristic evaluation function, representation and operators combine to lead search on a path from the current solution to a better one, hopefully to the goal state or a global optimum.

When operators induce small changes to the solution that result in small changes to solution quality, the *fitness landscape* is said to be *smooth*. The term *strong causality* describes the property where

small changes to the solution result in small changes to its fitness. This property can also apply to large changes in solutions. Strong causality indicates that there is a correlation between the size of solution transformation and the size of change in the resulting solution quality. Weak causality describes the case where there is little or no correlation between the size of the changes in solutions and fitness values. Fitness landscapes were suggested by Wright (1932) and later studied by Kauffman (1993) using the NK family of correlated landscapes. When the landscape is not smooth, local search methods are likely to get stuck with sub-optimal solutions.

In order to avoid sub-optimal solutions, metaheuristics use a variety of techniques to be robust to non-smooth, or rough, fitness landscapes. Most metaheuristic search strategies allow non-improving solutions to be selected during search. Two popular methods are *simulated annealing* (Kirkpatrick et al., 1983) and *tabu search* (Glover, 1986; Glover and Laguna, 1997). Simulated annealing is similar to a hill-climbing method that begins with a random current solution. However, in simulated annealing, a parameter (temperature) defines the likelihood that a worse scoring solution is accepted. Over time, the temperature is reduced, allowing the acceptance of worse solutions with smaller probability.

Tabu search generates the neighbourhood of a current solution and considers any of the new solutions that are not on a list of previously visited ones, the *tabu list*. Thus, the search is temporarily banned from re-visiting solutions, preventing cycling. In this way, if all improving solutions are on the tabu list, then this algorithm can also visit non-improving solutions. Tabu search can also contain an explicit method of directing the search toward similar or dissimilar new solutions, called *intensification* and *diversification*. Both simulated annealing and tabu search have been applied to a wide range of real-world problems (Pardalos and Resende, 2002; Glover and Kochenberger, 2003). Another class of metaheuristic search strategies are those based on the theory of Natural Selection (Darwin, 1859). These are described in the following section.

2.2 Evolutionary Algorithms

Darwinian evolution uses the principles of competition, inheritance, and variation within a population. These concepts are used to define a class of iterative improvement metaheuristic search methods. These methods, evolutionary algorithms, use a population of solutions and genetic operators to carry out search. Specifically, the evolutionary algorithm employs the following items:

- A population of candidate solutions called *individuals*,
- A *fitness function* that evaluates and assigns each individual a score, or *fitness value*,

- Transformation operators that produce *offspring* individuals from *parent* individuals, implementing the concept of inheritance through stochastic variation, and
- A stochastic selection method for selecting individuals with better fitness to produce offspring.

Given a population p at time t, the evolutionary algorithm applies variation operators v on the population. Variations are applied according to a selection method s, where individuals compete to be selected according to their fitness. A population p at time $t+1$ is found by:

$$p_{t+1} = v(s(p_t))$$

The transformation operators v provide new solutions by means of variations on existing ones. In evolutionary algorithms, these operators usually consist of *reproduction, recombination* and *mutation* operators. The population together with variation operators and competition provide a strategy for searching the space of solutions. Only those solutions which are reachable by the operators could be visited during the evolutionary process or *run*. Additionally, the use of a population in search, instead of searching with a single solution as in hill-climbing, allows for a parallel search of different areas of the search space representing alternative representations of similar fitness.

Evolutionary algorithms are often categorised into four main branches that are mainly distinguishable by their commonly used representation and operators: *genetic algorithms* use a bit-string and two-parent crossover, *evolutionary strategies* use a real-valued vector and Gaussian mutation, *evolutionary programming* employs a finite-state machine and mutation operators, and *genetic programming* uses a computer program or executable structure and two-parent crossover. These classifications represent common or initial implementations. Many implementations use components from different branches and make the classifications less accurate.

The genetic algorithm was originally represented by bit-strings in a fixed length genome. The term *genome* refers to the bit-string. Early work by Fraser (1957), reprinted in (Fogel, 1998), and Bremermann (1962) provided the foundations of genetic algorithms, which were later popularised by Holland (1975) and DeJong (1975). A bit-string requires a mapping to a representation that can be evaluated and assigned a fitness by the heuristic evaluation function. The genetic algorithm traditionally relied on various forms of crossover as the main operator. For instance, in one-point crossover with bit-strings a and b with bit positions indexed from $1 \ldots L$, a point p is chosen such that $1 \geq p \geq L - 1$. Next, the bit-strings exchange the portions of the string delineated by point p so that

$$\begin{aligned} &a = a_1 \ldots a_p a_{(p+1)} \ldots a_L \text{ and } b = b_1 \ldots b_p b_{(p+1)} \ldots b_L && \text{, becomes} \\ &a = a_1 \ldots a_p b_{(p+1)} \ldots b_L \text{ and } b = b_1 \ldots b_p a_{(p+1)} \ldots a_L && \text{after crossover.} \end{aligned}$$

Evolutionary strategies traditionally uses a direct encoding of the candidate solution, typically as a vector of real numbers (Rechenberg, 1965; Bäck et al., 2000a). The method began using a population consisting of a single individual. Offspring are produced by a Gaussian mutation operator and became the new population if they were better than the parent. Population-based evolutionary strategies uses two main selection schemes, the *comma* or *plus* strategy. In the (μ,λ) version, λ offspring replace the μ parents. In the ($\mu + \lambda$) version, parents and offspring compete to become the new set of μ parents. Evolutionary programming was originally based on the iterative adaptation of finite-state machines (Fogel et al., 1966). Finite-state machines are evaluated and assigned a fitness. A mutation operator is usually the only means of variation. In this sense, the population in evolutionary programming is like an ecosystem of species, where each individual is a different species.

Within the evolutionary algorithm framework, specialised methods such as multipopulation models (Cohoon et al., 1987), coevolutionary models (Potter and De Jong, 1994) and memetic models (Moscato, 1989; Krasnogor, 2002) exist to further blend evolutionary algorithms with other local search and heuristic methods. These *hybrid* algorithms typically combine global search strategies with local strategies, representing the state-of-the-art in optimisation (Davis, 1991; Pardalos and Resende, 2002; Glover and Kochenberger, 2003). Hybrids allow a general metaheuristic search strategy to be specialised for specific applications. Davis (1991) points out that the two objectives of generalisation and specialisation are conflicting in that increasing the performance of a method for a specific domain is likely to make the same method perform worse on other domains. The "No Free Lunch Theorems" by Wolpert and Macready (1997) emphasised this performance trade-off by explicitly stating that the average performance of all methods over all problems will be equal. However, this does not prevent some algorithms from being better than others on particular classes of problems.

Genetic programming is an evolutionary algorithm that represents solutions as programs. The remainder of this thesis will focus specifically on the canonical form of this method and representation.

2.3 Genetic Programming

Genetic programming is an evolutionary algorithm that uses either a procedural or a functional representation. This section describes this representation and the specific algorithm components used in the canonical version of the algorithm. The foundations of genetic programming are initially presented, followed by a discussion of the algorithm and a description of three common applications. Lastly, two important research issues and metaphors of genetic programming search are described.

2.3.1 Foundations

Genetic programming became a popular search technique in the early 1990s due to the work by Koza (1992). Angeline (1998) traced the historical foundations of genetic programming back to Friedberg (1958) and Friedberg, Dunham and North (1959) where iterative random changes made to computer programs induced better programs. Learning machines were proposed earlier by Turing (1950), where an automated process similar to evolution in combination with human interaction was thought to be a possible way for programs to acquire intelligent behaviour. The form of genetic programming used today is most closely related to work done by Cramer (1985), where a tree representation of programs was used in conjunction with subtree crossover to evolve a multiplication function. This work also employed the use of partial credit assignment for function activity, input dependence and iterative constructs, issues that are not traditionally considered in canonical genetic programming. Other background and foundational research of genetic programming and evolutionary algorithms can be found in (Banzhaf et al., 1998; Fogel, 1998; Bäck et al., 2000a; Koza, 1994). More modern approaches to genetic programming and evolutionary algorithms are described in (Langdon and Poli, 2002; Bäck et al., 2000b; Koza et al., 2003). Some of these approaches are discussed in later chapters.

The theoretical foundations of genetic programming are summarised in Langdon and Poli (2002). Theories of evolutionary algorithms use abstract representations of the solution space, called schemata, to describe various components and behaviours of the algorithm. Holland's (1975) notion of schema for genetic algorithms was extended by Koza (1992) to include syntax trees. Syntax trees are the common representation in genetic programming, and these schema were intended to represent trees that have common subtrees. In this case, subtrees represent functional code that is combined over time to create better programs (Altenberg, 1994; Rosca and Ballard, 1996). O'Reilly (1995) extended the idea of schema for genetic programming trees to allow for subtrees to contain wildcards, giving a more flexible definition of schema. Rosca (1997) defined a similar schema based on rooted trees, also using wildcards to allow for schema to represent templates.

Theories based on parse tree schemata are often complex. This is mainly due to the many random decisions in the algorithm, requiring the modelling of many stochastic events. However, it is not always clear that schemata are selected and combined in a predictive way. That is, the fitness resulting from the combination of two schemata with high fitness is often unpredictable. That does not, however, lessen the importance of the development of theoretical models, which have produced many practical results. The rooted tree schema (Rosca, 1997b) provided an improved understanding of the effects of popular operators and representation, while Poli (2003) used theoretical results to justify a method to control growth in solution size. Langdon and Poli (2002) and Poli and McPhee (2003a,2003b) describe their recent developments of exact schema theories for genetic programming using a complete and exact description of the representation and operators. As

noted in (Langdon and Poli, 2002), this approach is costly to compute since, in a sense, it simulates the actual execution of the algorithm with all its complexities and degrees of freedom. Thus, to know what exact schemata will be generated, one can either compute an enormous number of probabilities of events and their likely results, or one can execute the algorithm.

2.3.2 The Genetic Programming Algorithm

Genetic programming adopts a similar search strategy as a genetic algorithm, but uses a program representation and special operators. It is this representation that makes genetic programming unique. The basic algorithm is as follows:

1. Initialise a population of solutions
2. Assign a fitness to each population member
3. While the Stopping criterion is not met
4. Produce new individuals using operators and the existing population
5. Place new individuals into the population
6. Assign a fitness to each population member, and test for the Stopping criterion
7. Return the best fitness found

The rest of this section describes the components of this algorithm in more detail, beginning with the representation.

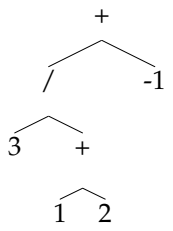
2.3.3 Representation of Solutions

The original formulations of genetic programming considered Lisp S-expressions as candidate solutions. The Lisp programming language is popular in artificial intelligence research as it was designed for symbolic processing, see (Shapiro, 1990) for a further discussion. S-expressions, or symbolic expressions, are the basic objects in Lisp and are naturally represented as syntax trees, where leaves represent terminals (variables or constants) and nodes represent functions. The arity of a node is the number of arguments it expects to receive. In genetic programming, to overcome typing issues related to passing arguments and function values to functions, it is standard to use a *strongly-typed* system (Montana, 1995) where variables and functions have the same type. Grammar-based genetic programming systems can use multiple types easily, but require a defined grammar and specialised operators, e.g. (Whigham, 1995; Ryan et al., 1998).

The evaluation of a syntax tree is performed as a depth-first walk of the tree. Before evaluating a node, each of its arguments must first be evaluated. Thus, one may think of the syntax tree evaluation algorithm as a recursive call on the root node of the tree, which in turn evaluates each of its children, typically from left to right. Function and terminal nodes return their values up the tree to their parent, where terminals can only return their value. In the domain of mathematical functions, the expression $((3/(1+2))+(-1))$ can be represented by an S-expression in prefix as:

$$(+\ (/\ (3\ (+\ (1\ 2))))\ -1),$$

which can then be represented as a syntax tree as:



The representation of candidate solutions in genetic programming is not limited to single syntax trees. Auxiliary data structures such as memory arrays are also used (Spector and Luke, 1996), as are data structures containing several syntax trees to represent one solution (Luke, 1998). The encapsulation of functions has been accomplished with *automatically defined functions* (Koza, 1994) or *automatically defined macros* (Spector, 1996). More recent advances have seen "architectural-altering" operators, loops and recursion (Koza et al., 1999). Additionally, when using functional languages such as ML in a genetic programming framework, iteration and recursion can become more natural (Olsson, 1995).

2.3.4 Initialisation

The genetic programming algorithm requires an initial population of individuals, or syntax trees. There are many possible methods to perform tree initialisation. The ramped half-n-half method is the most commonly used. It was introduced by Koza (1992) and probabilistically selects between two recursive tree generating methods: Grow and Full. The Grow method choses a depth d and randomly picks functions or terminals to build a tree, no deeper in any branch than d. The Full method is the same but only choses functions until d is reached, creating full trees of depth d.

Several methods have been used to create different distributions of initial trees, where the general consensus is that a more uniform and random distribution is better for the evolutionary process. These methods have recently been described in (Luke, 2000; Luke and Panait, 2001). While the Full and Grow methods were probably initially (and subsequently) used due to their simplicity,

they offer little control over tree creation. However, while other methods do give better control, e.g. (Iba, 1996), it is not clear what methods are better for different situations and whether the complexity of these methods are justified by their performance. Thus, in the research presented in this thesis, the ramped half-n-half method will be used for initialisation.

2.3.5 Fitness and Selection

Genetic programming requires an evaluation, or fitness, function to determine the quality of an individual. The fitness value is typically a single scalar value that allows selection methods to distinguish between various levels of individual quality. Multiobjective methods often use a vector to represent fitness. Most of the computation time, particularly for more complex problem domains, is spent in the fitness evaluation. For instance, in a robotic control problem, each individual might represent an algorithm to move a robot from one location to another. Each individual must then be run for the purpose of fitness evaluation in a simulator or on an actual robot, where both tasks may take considerable time.

To determine a fitness value, the individual is typically applied to a fitness case, or set of fitness cases. The score on the fitness case(s) is represented in the final fitness function value. For example, in the above robotic control problem example, executing a program that controls a robot in a simulator would represent a fitness case. The fitness value assigned to the robot on this fitness case could represent the distance from some predefined location and the final location of the robot. If this process was repeated several times by starting the robot in different initial locations, the values returned by these fitness cases could be averaged to represent the fitness value assigned to the individual. In the problem domains described later in this chapter, a fitness value will sometimes be the result of applying a program to one fitness case and sometimes it will be the result of applying the program to a set of fitness cases.

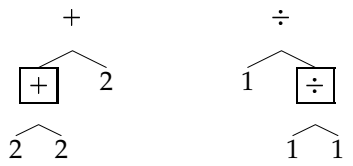
The design of a good representation and fitness function is critical. Ideally, the representation and fitness should have the property of *strong causality* where small changes in the genotype represent small changes in fitness (Rosca and Ballard, 1995). However, weak causality, the opposite of strong causality, is more common in genetic programming. Weak causality implies more "instability" of the programs evolved by genetic programming, where small changes to solutions often yield large changes in fitness.

While fitness functions are designed to distinguish between individual quality, code growth and other dynamics can also be controlled with components added to the fitness function (Luke and Panait, 2002b). Dynamic fitness functions and hierarchically defined fitness functions (Langdon and Poli, 2002) are examples of other ways to guide the search.

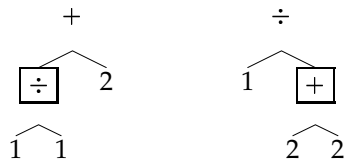
To create the next population of individuals in a generational algorithm, individuals are probabilistically selected from the current population based on fitness. The most common method in genetic programming is *tournament selection*. Tournament selection chooses t individuals from the current population and returns the one with the best fitness. Typically, one tournament is held for each individual participating in crossover, mutation or reproduction. Other forms of selection consist of truncation selection, fitness-proportionate selection, ranked selection and selection based on multiple criteria (for example, selecting individuals based on fitness and then on size). See (Blickle and Thiele, 1995) for a discussion of selection methods.

2.3.6 Recombination, Mutation and Reproduction

Recombination is usually implemented as subtree crossover between two parents. The mechanics of the subtree crossover method were initially described by Cramer (1985) and Koza (1992) and have been examined in detail in a variety of studies, e.g. (D'haeseleer, 1994; Luke and Spector, 1998; Langdon, 2000b; Igel and Chellapilla, 1999; Gathercole and Ross, 1996). The algorithm can be described as follows: Two trees are selected from the population, a subtree in each tree is selected and the two subtrees are exchanged between the trees. Either one or both children are considered for the new population. For example, consider the two binary trees with roots $+$ and $\div$,



Two crossover points are chosen ($\boxed{+}$ and $\boxed{\div}$) and subtrees are exchanged to produce the two following trees:



Subtree selection is done by assigning a uniform probability to all internal nodes and leaf nodes separately. Then, an internal node selection probability, usually set to 0.9, defines the frequency of leaves or subtrees selected for recombination. Note that, typically, as trees grow in size, the probability of selecting subtrees near the leaves grows. This is because there are *more* nodes in these locations, giving them a higher cumulative probability of being selected.

Since in canonical genetic programming all functions and terminals return and expect the same type, any exchange of subtrees between two trees will be valid. Many possible variations of recombination exist. For example, Homologous and Size Fair crossover (Langdon, 2000b) attempt

to preserve tree structures and the size of exchanged subtrees. Other operators are described in (Langdon, 1998a).

Subtree crossover tends to be the dominant operator in genetic programming, while mutation operators are often used at lower rates. Subtree mutation was investigated in comparison with subtree crossover in (Luke and Spector, 1998). Other forms of mutation include single-node mutations and various forms of code-editing to remove unnecessary code from trees. The reproduction operator copies an individual from one population into the next.

The production and selection of new individuals is carried out in a *generational* or *steady-state* algorithm. In a generational algorithm, a new population is created from parents in the current population. In a steady-state algorithm, one offspring is created and placed into the existing population, either randomly or based on fitness. The *comma* and *plus* methods for offspring and parent selection used in evolutionary strategies are also possible, see Section 2.2. Other examples of offspring production force children to compete amongst themselves, such as brood selection (Altenberg, 1994; Tackett, 1994), or use methods based on similarity to bias toward similar or dissimilar parents, such as disassortative mating (Ryan, 1994).

2.3.7 Stopping Criterion

A *generation* consists of the production of a new population in a generational algorithm. A similar definition is used for a steady-state algorithm. A maximum number of generations usually defines the stopping criterion in genetic programming. However, when it is possible to achieve an *ideal* fitness, this can also serve as the stopping criterion. In this thesis, a maximum generation number is used, regardless of whether the ideal fitness is achieved. Other criteria are possible, such as a measure of diversity loss or a lack of fitness improvement.

2.4 Application Domains

To further understand genetic programming, it is useful to consider the problem domains which were used to motivate the field and to develop a theoretical understanding of it. Such problem domains are also employed to test new representations and operators. These standard problems are used throughout the genetic programming literature and were introduced by Koza (1992) and later used by other researchers, e.g. (Daida et al., 2001; Luke, 2001; Luke and Spector, 1998; McPhee and Hopper, 1999; Soule and Heckendorn, 2002). This section surveys these problems to introduce the domains which are used in this research.

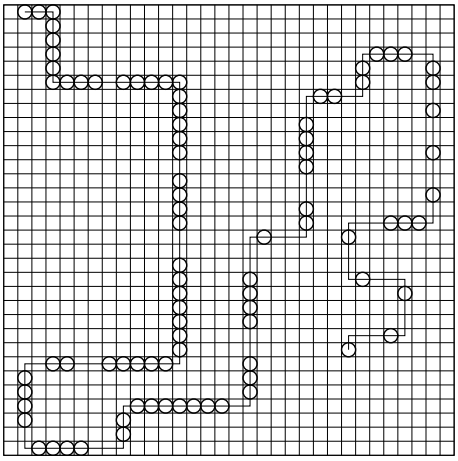


FIGURE 2.1: The Sante Fe Trail for the Artificial Ant Problem. Food pellets are denoted with a circle in the above grid.

2.4.1 Artificial Ant

The artificial Ant problem was popularised by Koza (1992), but was originally developed for the field of *artificial life* by Jefferson et al. (1991). The Ant problem consists of finding the best strategy for picking up food pellets along a trail in a grid. The solution to the problem is an algorithm for collecting food. The Sante Fe trail is often used for the Ant problem. The Sante Fe trail consists of 89 food elements on a two dimensional, 32×32, toroidal grid, shown in Figure 2.1. The 'ant' starts in the North-west corner, facing East.

Ant Functions and Terminals

The functions and terminals are executed to move the ant on the grid during evaluation. The typical functions include:

- *if_food_ahead(a,b)* : if the 'ant' faces a food pellet then evaluate *a*, else evaluate *b*,
- *progn2(a,b)* : evaluate *a*, then *b*, returning *b*'s value, and
- *progn3(a,b,c)* : evaluate *a*, *b* then *c*, returning *c*'s value.

The last two functions are simple sequencing functions. The following terminals provide the 'ant' with locomotion and change of direction:

- *left* : turn the 'ant' left by 90 degrees,
- *right* : turn the 'ant' right by 90 degrees, and

- *move* : move the 'ant' forward by one square/step.

Ant Fitness Assessment

The fitness for this problem is measured as the number of pellets missed out of the total on the trail. The ant is typically given a total of 600 time steps[1] to collect food pellets, where each terminal takes one time step to execute. Each candidate program is re-evaluated, without re-initialising the 'ant', until the all the food has been collected or the maximum number of time steps is reached.

Ant Related Studies

The Artificial Ant problem was investigated in numerous studies. Langdon and Poli (2002) examined several aspects of the search space for the Ant problem. The authors measure the search effort required for genetic programming, several variations of genetic programming, several random search techniques and several stochastic search methods. While genetic programming performed at least as good as the random search methods, some versions of stochastic hill-climbing, including population based hill-climbing, performed equally well or better than genetic programming. Langdon and Poli noted that the Ant problem is highly deceptive. That is, there are many possible solutions that are symmetrical, and because there is no requirement that a particular path is followed, solutions with equal fitness can take very different approaches to solving the problem. Langdon and Poli later showed that by encouraging the population to follow similar paths, the problem became considerably easier for genetic programming.

2.4.2 Even Parity

Machine learning is typically concerned with inducing a model that can correctly classify input and output pairs. The set of boolean problems provides a similar framework for genetic programming. Langdon and Poli (2002) provide a discussion of boolean program spaces, which is also related to the field of evolvable hardware and circuit design (Koza et al., 1999).

Within the class of boolean problems, the Even-n-Parity problem requires the correct classification of bit strings of length n having an even number of 1's. For example, Figure 2.2 shows the truth table for Even-3-Parity, where an even number of 1's is classified as "True". In this thesis, the case of $n = 5$ will be considered.

[1] Koza (1992) reported the number of 400, but work by W. Langdon suggests this number was 600.

D1	D2	D3	Even-Parity
0	0	0	True
0	0	1	False
0	1	0	False
0	1	1	True
1	0	0	False
1	0	1	True
1	1	0	True
1	1	1	False

FIGURE 2.2: Truth table for the Even-3-Parity problem.

Parity Functions and Terminals

Logical functions are used to determine parity in the binary bit strings. The typical functions used include:

- *and(a,b)* : returns 1 if *a* and *b* are 1, else 0,
- *or(a,b)* : returns 1 if *a* or *b* are 1, else 0,
- *nand(a,b)* : returns 0 if *a* and *b* are 1, else 1.
- *nor(a,b)* : returns 0 if *a* or *b* are 1, else 1.

In the Parity problem, the terminals representing the n bits are denoted as:

- *D1,...,Dn*.

Parity Fitness Assessment

The Even-5-Parity fitness function is the number of wrong classifications out of the 2^5 possible bit strings of length 5.

Parity Related Studies

The Parity problem has also been investigated in detail in (Langdon and Poli, 2002). The most characteristic aspects of this problem are the few test cases that define fitness and the requirement that all terminals are processed by a solution to achieve the ideal fitness. Additionally, because of the nature of the problem, a random guess on all fitness test cases will lead to a score of 50%. O'Reilly and Oppacher (1994,1995) performed a comparison of several hill-climbing methods using similar

genetic programming operators to show that hill-climbing generally performed better than genetic programming on the 6 and 11 Multiplexer problem. Similar results were also obtained in (Juels and Wattenberg, 1995). While the two problems, Multiplexer and Parity, are not identical, they share several similar features (e.g. boolean functions and terminals and very few fitness function values).

2.4.3 Symbolic Regression

In many scientific disciplines, finding a model explaining the relationship of large and complex sets of data is required. However, the task is often difficult and many methods have been employed to uncover models of data. The domain of symbolic regression is one method where input and output pairs are used to infer a functional model, typically consisting of a function and its coefficients. Keijzer (2001) researched the application of genetic programming on the Regression domain in the context of "scientific discovery", demonstrating several ways to improve its application.

The Regression problem attempts to find a program that approximates a target function. A target function $f(x) = y$ is applied to domain values, x, in a pre-determined range. The resulting y values are then compared with the candidate program's value upon the same x values. This thesis uses the Quartic polynomial,

$$f(x) = x^4 + x^3 + x^2 + x,$$

the Rastrigin function,

$$f(x) = 3.0n + \sum_{i=1}^{n} x_i^2 - 3.0\cos(2\pi x_i),$$

the Binomial-3 polynomial,

$$f(x) = (1 + x)^3,$$

and several random polynomials (described in Chapter 5). For the Rastrigin instance, x is in the range $[-5.12, 5.12]^n$, for the Quartic, Binomial-3 and random polynomial instances, x is in the range $[-1.00, 1.00]$. The number of (x, y) pairs is 20 for both the Quartic and Rastrigin instances and 50 for the Binomial-3 and random polynomial instances. Figure 2.3 shows the Quartic polynomial and the Rastrigin function ($n = 1$).

Regression Functions and Terminals

Regression problems use mathematical functions to approximate a target function. The division and logarithm function are usually "wrapped": division returns 1.0 if the denominator is equal to or very close to 0, while logarithm returns 0 if its argument is equal to or very close to 0. Additionally, it is common, especially for functions with coefficients such as the Rastrigin, to use ephemeral

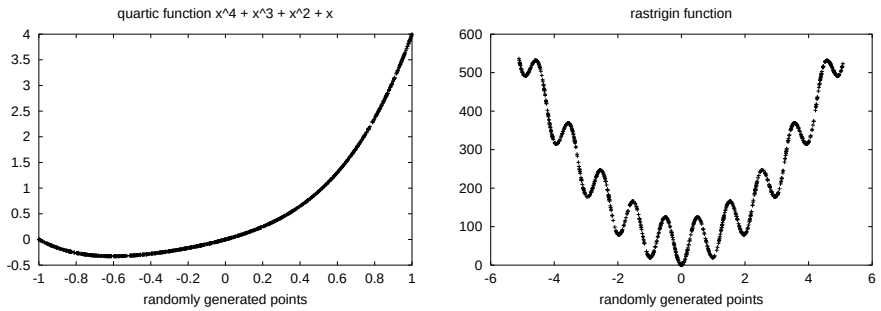


FIGURE 2.3: The Quartic and Rastrigin functions.

random constants (ERCs). ERCs are constants that keep their value after initialisation for the remainder of the run. In the random polynomial instances in Chapter 5, the ERCs are in the range of [-1,1] while they are varied for the Binomial-3 instances. The experiments that use the Rastrigin function in Chapter 4 do not use ERCs.

The Regression domain functions are usually a subset of the following:

- *a + b, a - b, a * b* : return addition, subtraction and multiplication of their arguments, respectively,
- *a / b* : protected division, returns 1 if denominator is ≈ 0, otherwise returns the quotient,
- *sin(a), cos(a)* : returns sine and cosine of the argument,
- *log(a)* : returns 0 if a ≈ 0, else the logarithm of the argument,
- *exp(a)* : returns e^a.

The following terminals are common in the Regression domain:

- x : from the pair (x, y),
- *ERCs*.

Regression Fitness Assessment

Fitness for the Regression problem instances is determined by summing the squared difference between the target value for each x value. In Chapter 5, fitness is reported as "adjusted fitness", where this is defined by the following equation:

$$\text{adjusted fitness} = \frac{1}{1 + \text{fitness}}.$$

Regression Related Studies

Many studies have used the Regression problem domain as a testbed. The problem has been characterised by both Daida et al. (2001) and O'Reilly (1998) as being unique in that it contains many conflicts of context and content in the representation. Keijzer (2003) found improvements using interval arithmetic and a simple fitting method, and other improvements were found using more advanced models for symbolic regression of polynomials (Nikolaev and Iba, 2001).

2.5 Scalability and Fitness Landscape

Two complex issues in genetic programming is the ability to scale toward harder and larger problem instances and the landscape induced by the representation, operators and fitness function. This thesis approaches both of these issues by focusing on the population used by genetic programming.

There are many elements of genetic programming which limit the scalability of the algorithm. First, evaluating a solution can be computationally expensive. While evaluating a single solution alone is not a problem, a very large population of solutions easily becomes a problem. A bias toward the increase in solution size compounds this problem by increasing the evaluation time and the memory used by the population. Also, it is not clear that genetic programming can be applied effectively to when the combinatorial search space is increased, nor is it clear that very simple implementation decisions, such as the population size, do not drastically change the difficulty of a problem instance.

A second major issue in genetic programming is the basic choice of representation, operators and fitness function. The elements define the fitness landscape. The representation of solutions by computer programs is straightforward for modelling a variety of problem domains. However, transforming one program to another, while maintaining some commonality of behaviour between the two is not, i.e. the property of strong causality. Thus, when the representation, operators and fitness function have the property of weak causality, the fitness landscape can be described as *rough*. Not only has the operator in genetic programming come under scrutiny, but several representation variations have been proposed to improve the algorithm, e.g. grammar-based genetic programming (Whigham, 1995) and linear genetic programming (Keller and Banzhaf, 1996).

Related to the issue of strong and weak causality in genetic programming is the issue of *context* and *content* in the representation and fitness function. The content of a syntax tree refers to its functions and terminals. The context of a group of functions and terminals in a particular syntax tree is the way in which these primitives contribute toward fitness, and is typically determined by the other elements in the tree. However, moving this content, or group of functions and terminals,

to another syntax tree may change its context. Thus, selecting a portion of one solution that has good fitness and placing it in another solution with good fitness is not guaranteed to preserve the original context of the content. The issue of content and context in the genetic programming system is likely to effect the causality of the system.

Both of the previous two issues, scalability and fitness landscape, have strong links with the population and its dynamics. For instance, if the size of the population could be reduced without cost to performance, computation could be drastically reduced. As the population has a strong effect on guiding the search in the solution space, the issues of scalability and landscape roughness could be addressed to a degree by different population dynamics. To better understand genetic programming and related issues, this thesis will address population dynamics by studying the more specific issue of population diversity. Diversity describes the amount of variation in the population, and as shown in later chapters, has wide-reaching effects on many aspects of the search process.

2.6 Metaphors of Search

Genetic programming is an *evolutionary* algorithm. The assumed metaphor for this type of search is that it proceeds by adaptation: solutions adapt or change to meet predefined objectives. However, when this metaphor of adaptation is applied to the standard genetic programming algorithm, it is somewhat misleading and too general. It is misleading because canonical genetic programming, due to structure and content convergence, has only a limited ability to *adapt* during the evolutionary process. That is, it is likely at some point to be unable to change solution quality. The metaphor is too general as genetic programming is usually applied to search problems where a component is the optimisation of a specific objective. However, the concept of adaptation seems to imply that a solution can equally adapt to one or more objectives, as is typically the case in Nature. It is likely that a metaphor exists which is more specific than *evolution* and is still accurate.

One such metaphor that was previously proposed is that of Beam Search (Tackett, 1994). Beam search is similar to the best-first heuristic but only allows a limited capacity of memory of previously visited solutions. Thus, portions of the search space may be forgotten and optimal solutions may be missed. In genetic programming, the population and selection methods provide the basis for the metaphor. These elements provide a kind of memory of previous search and the ability to select the most promising aspects to continue search with.

Another possible metaphor is that of hill-climbing. Hill-climbing is a memory-less version of beam search. While comparisons are made between genetic programming search quality, hill-climbing (O'Reilly and Oppacher, 1994) and random search methods (Langdon and Poli, 2002), to characterise the search behaviour is a different matter. It is clearly not the case that genetic programming

is a *random* search. The population, selection and variation of better solutions produce a search by the trial-and-error of new solutions, but a metaphor of random search would suggest that the search proceeds with no direction from previous solutions. Hill-climbing, on the other hand, may be an applicable metaphor as it captures the sense that while a population does exist (providing the memory component for the Beam-Search metaphor), convergence and operator behaviour are likely to prevent the population from returning to a previous state.

It may be the case that beam search and hill-climbing metaphors offer an optimistic and pessimistic metaphor of genetic programming search, respectively. A variation of hill-climbing, stochastic hill-climbing (Juels and Wattenberg, 1995), seems particularly appropriate as it considers neutral moves on the fitness landscape. In the following research, a metaphor of search is sought to explain results and algorithm behaviour. In Chapter 5, a metaphor of hill-climbing appears to agree with the results and with much previous literature. In light of the complexity of genetic programming and other evolutionary algorithms, such metaphors of search may lead to new theoretical models.

2.7 Summary

This chapter has described search, evolutionary algorithms and the genetic programming algorithm and representation. Three common problem domains were introduced. Lastly, two open research issues, their relationship with the population, and metaphors of search relating to genetic programming were presented.

The procedural representation and evolutionary search strategy used by genetic programming makes it an intuitive algorithm for automatic programming and search. However, the issues of scalability and landscape roughness can present serious issues to its applicability. These issues have received much attention from the community. Instead of proposing new solutions to these problems, the aim of the remainder of this thesis is to develop a sound working knowledge of population dynamics from which new solutions can then be developed. To achieve this goal, the research will focus on the issue of population diversity. Once this is achieved, many new possibilities for improving search with genetic programming will become intuitive. Toward this goal, three key questions are outlined in the next chapter which are addressed in the following experimental studies.

CHAPTER 3

ISSUES IN GENETIC PROGRAMMING

The genetic programming literature often cites the importance of maintaining diversity to avoid premature convergence toward local optima (Ekárt and Németh, 2000; McKay, 2000; McPhee and Hopper, 1999; Rosca, 1995a; Ryan, 1994). What does diversity mean to a population-based search algorithm? Certainly, variation is required in the principles of biological evolution. While genetic programming is not the same process that occurs in Nature, there are many different types and possible levels of diversity.

At the center of this thesis are three questions that are related to diversity. By exploring these issues, a clearer understanding of genetic programming and diversity will be achieved. Each of these three questions motivates the research that follows. The questions are:

1. *How can diversity be measured and controlled, and are there ideal levels of diversity?*

   The numerous references to the culpability of 'the loss of diversity' for poor results, e.g. as suggested by the several methods designed to increase diversity discussed in Chapter 4, suggests that the prevention of this loss will somehow improve results. However, it will be shown later that diversity has been measured and used in a variety of ways that are often conflicting.

2. *Genetic programming is a population search method, thus, what effect does population diversity have on other aspects of the search process?*

   One challenging problem in genetic programming is the increase of solution size that is not accompanied by an increase in solution quality. While the literature describes the mechanics that cause growth, e.g. (Langdon and Poli, 1998a; Luke, 2003; Soule and Heckendorn, 2002), few have argued as done in this thesis that the varied rate of growth is an artifact of population diversity.

3. *What specific role does diversity play in the evolutionary process, i.e. do dissimilar individuals contribute offspring differently than the rest of the population?*

   The population provides the main reservoir of genetic material from which to produce new solutions. The population is expected, to a degree, to simultaneously occupy different parts of the search space. However, to provide good solutions, it is accepted that the population converges toward one optimum. These two possibly conflicting functions of the population are important to understand, particularly in context of diversity.

The purpose of this chapter is to examine these issues in more detail.

3.1 Diversity Measures and Methods

Diversity is not the goal of evolutionary algorithms. Ideally, as in nature, diversity would be a side-effect of the representation and operators that, when the right or sufficient level is achieved, encourages good performance. While population initialisation in genetic programming does not usually allow duplicate individuals, future populations are usually not bounded by such a constraint. In fact, duplication of individuals is common and often explicitly promoted by more elitist methods and the reproduction operator.

A natural conclusion as to the cause of run failure is the wrong level of diversity, specifically too little diversity. However, this idea is problematic for several reasons. Genetic programming is not typically implemented as an 'open-ended' evolutionary system. The algorithm is said to 'converge' when it is unable to find new solutions or improve solution quality. Without explicit pressure, genetic diversity and the ability to make variation and improvement will be lost during the evolutionary process. Knowing whether this will occur in the earlier or later stages is difficult. As fitness is a function of the syntax tree representation, a loss of different syntax trees also causes a loss of different fitness values. How does one know the right kind and level of diversity required to produce quality solutions?

3.1.1 Measuring Diversity

Genetic programming typically employs a syntax-tree representation, often using binary trees. A population of trees exists in each generation. Function and terminal sets consist of anywhere from 1 to several terminals and usually 3 or more functions. The number of different binary tree shapes with n internal nodes can be found using the Catalan number $C_n = (2n)!/(n!(n+1)!)$, (Lucas et al., 1993). The beginning of this series, where each term describes the number of binary tree

configurations with n internal nodes, i.e. trees of size $2n + 1$, consists of the following terms:

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, \ldots$$

Now, for every tree of size $2n+1$, the total number of unique trees given the function and terminals sets creates an enormous space including $C_n \times f^n \times t^{(n+1)}$ programs, where f is the number of functions and t is the number of terminals. Within this space there may be many equivalent trees, functionally and based on the fitness function. What aspect of tree shape, content or behaviour should diversity measures capture? What are the critical elements? Should measures concentrate on the node level, i.e. based on the number and types of function and terminal nodes (McPhee and Hopper, 1999), or should more problem specific measures be used that describe detailed solution behaviour (D'haeseleer and Bluming, 1994)?

The term *diversity* is often used without definition. The implicit assumption is that it refers to the solution representation. However, it is useful to also consider aspects of solution behaviour or fitness. In the following research, the term *genotype* refers to the shape and content of the solution, while *phenotype* refers to the fitness value. The term *phenotype* generally eludes to a more descriptive measure of solution behaviour. However, the standard practice in the literature uses *fitness value* and *phenotype* interchangeably. Later, a more complex definition of solution behaviour will be used that describes solutions in more detail than typical fitness functions.

Based on the type of diversity being measured, several measures of diversity can be defined. The number of unique genotypes was the traditional diversity measure (Koza, 1992; Langdon, 1998a). The number of unique fitness values was also considered a diversity measure (Rosca, 1995b). There are obviously many other possible definitions. Some of these definitions are described in Chapter 4.

3.1.2 Controlling Diversity

Is there an ideal level of a particular type of diversity? The two recognised phases of the evolutionary process, exploration followed by exploitation (Eiben and Schippers, 1998), make this question difficult to answer. Without knowing *a priori* the existence of a type and ideal levels of diversity, blindly controlling diversity in a population seems problematic. Instead, two different diversity strategies may be required: a diversification and an intensification strategy, similar to those explicitly defined in tabu search (Glover and Laguna, 1997). A related method was recently defined in genetic programming based on edit distance and adaptive control based on fitness improvement (Ekárt and Németh, 2000). However, the exploration and exploitation aspects of search are not the only issues to consider when attempting to control or modify population diversity. A fuller discussion of methods used to control diversity is presented in Chapter 4.

3.2 The Effects of Population Diversity

Genetic programming works by selecting individuals and recombining their genetic material to produce new offspring. It is common for genetic programming to use only the subtree crossover operator. When mutation operators are used, their effects are not unlike crossover and they are often used at low rates. The type of individuals in the population will obviously affect the type of individuals selected and the type of offspring produced. Recombining similar or dissimilar individuals will effect offspring size and fitness. Diverse and non-diverse populations can make the search more difficult depending on the landscape defined by the representation, operator and fitness function. The possibility of context and content conflicts in the representation can further complicate the effects of population diversity. Also, population diversity can effect selection pressure, making the search more random or more focused.

3.2.1 Code Growth

Code growth is a well-documented phenomenon of evolutionary systems using a variable-length representation, as is used in genetic programming. When an increase in solution size does not correspond with fitness improvement then computation time is wasted and the readability of solutions is decreased. Several recent studies of the causes of code growth exist. Luke (2003) presented a theory based on the depth of modification points and Soule and Heckendorn (2002) assessed the viability of three hypotheses: the protective, the drift and the removal bias. There are two major contributing factors to code growth in genetic programming emphasised in these studies:

1. The smaller the removed portion of a tree the less likely that the fitness of the resulting offspring will be worse than the parent's according to the *removal bias* (Soule and Heckendorn, 2002; Igel and Chellapilla, 1999).

2. Child survivability is linked to deeper nodes selected for modification in their parents according to the *depth-based theory of code growth* (Luke, 2003). This gives a bias toward larger parents which will tend to produce larger children.

These two points are overlapping, and are also somewhat consistent with other theories of code growth, specifically Langdon and Poli's *fitness causes bloat* theory (Langdon, 2000a; Langdon and Poli, 1998a; Langdon et al., 1999), which suggests that the space of larger trees contains more better fit programs. As fitness increases or stays the same with tree size growth, the selection of crossover points nearer to the leaves increases, the size of the removed portion decreases and the modification points are deeper. Thus, as trees grow, their modification points become deeper, their removed

subtrees become smaller and their children's survivability is higher. There is a clear bias toward a constant rate of growth during evolution.

However, empirical evidence also shows a wide range of code growth rates that these theories do not predict. In a series of random runs, genetic programming may produce a wide range of solution sizes. For example, Figure 4.5 in Chapter 4 shows the average size of an individual in each population for 50 random runs. In each of the experiments, a wide range of sizes are produced. Understanding the mechanics of the population that allowed for smaller solutions and the opposite mechanics that caused larger solutions seems paramount to controlling code growth while maintaining fitness improvements. If solutions are predisposed toward growth due to the recombination of tree types, controlling population diversity could allow better control of code growth. Also, understanding the effects and causes of different levels of population diversity could allow a more thorough description of the mechanisms that cause growth.

3.2.2 Problem Difficulty

Another issue in genetic programming is how to identify important properties of problems or problem instances. Specifically, what makes a problem difficult with respect to genetic programming? Daida et al. (2001) investigated a tunable problem, the binomial-3 function with varying ephemeral random constant ranges, to show that difficulty can be increased without changing the combinatorial search space. In this case, genetic programming difficulty increases with the increased range of ERC values. The authors suggested that the conflict between content and context is largely responsible for increased difficulty. O'Reilly and Goldberg (O'Reilly, 1998; Goldberg and O'Reilly, 1998) used constructed problems to highlight the content and context dependencies in genetic programming solutions. The authors investigated how partial solutions contribute toward fitness and how they make solving the problem more difficult (O'Reilly and Goldberg, 1998). Recent research has also investigated possible *structural mechanisms* that make search more difficult (Daida, 2002; Daida et al., 2003b). Population diversity, among other things, could be a way to reduce problem difficulty by filtering misleading or deceptive solutions from the population and search.

3.2.3 Selection Pressure and Deception

Selection methods select fitter individuals to produce new individuals. The fitness function defines a scalar value for each individual that the selection method uses to compare individuals. However, the loss of different fitness values in the population leads to the reduction of selection pressure on individuals with the same fitness. For example, if the population contains only one fitness value, then a selection method based on fitness will become random. When fitness-based diversity is

low, particularly among the fitter individuals, selection and search become more random. Previous work has also linked selection pressure to code growth (Smith and Harries, 1998; Langdon and Poli, 1998a; Tackett, 1994).

Low phenotype diversity in the population can also represent another property of the population, *deception*. Goldberg (2002) provides a discussion of deception in context of genetic algorithms and building blocks. Langdon and Poli (1998b) describe deception in terms of the Artificial Ant problem for genetic programming. 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. Or deception can occur when solutions that have relatively good fitness are not amenable to further improvement. Selecting these individuals would not lead to better solutions. In both cases, the selection method does not have enough information about the problem or population to avoid deceptive solutions.

3.3 The Role of the Population

Fitness evaluation is typically responsible for the majority of computation time in genetic programming. Bigger populations require more computation. Understanding clearly the role of the population during search is likely to allow the controlled reduction of population size and improved efficiency. Additionally, understanding this role is also likely to allow populations to be evaluated on their potential for providing good search. This section examines several issues related to the role of the population during search.

3.3.1 Local or Global Search

Metaheuristic methods, such as genetic programming, are typically considered as global search methods. However, the locality of the operator, subtree crossover, has led the search to be characterised as more of a local search method (Poli and Langdon, 1998a; Langdon and Poli, 2002). Research has shown that the loss of genetic diversity in populations is often followed by small fitness improvements, likening the search to a blind random search on the converged population (Gathercole and Ross, 1996; McPhee and Hopper, 1999). Genetic programming is then described as "behave[ing] like a set of parallel stochastic hill-climbers" (Poli and Langdon, 1998a) as it has a limited ability to search much of the space of possible solutions.

Without diversity in the population, genetic programming is more likely to be capable of only sampling close genetic neighbours. When this behaviour is repeated over several random runs, the runs appear to be stochastic hill-climbers. While the loss of diversity could encourage improvement by

performing a more local search around the best solution found so far, too much early convergence is likely to make the population more susceptible to local optima (McPhee and Hopper, 1999).

Genetic programming was compared with hill-climbing methods using similar representations and operators (O'Reilly and Oppacher, 1995; O'Reilly and Oppacher, 1994; Langdon and Poli, 2002; Langdon, 1998b; Juels and Wattenberg, 1995). Different problems were solved better and worse using genetic programming. While these studies will be referred to later to help explain experimental results, they also show how simple algorithms that are made to behave like genetic programming can sometimes provide good and better results. For instance, in (Juels and Wattenberg, 1995), a hill-climbing strategy is amended to allow neutral moves, called stochastic hill-climbing, which seems appropriate for genetic programming. Standard hill-climbing typically accepts only solutions that are improvements to the current solution.

A high level of convergence in genetic programming suggests a diminished exploration ability, and the comparisons to hill-climbing methods suggest that these two methods sometimes behave similarly. Should genetic programming be considered as more of a global or a local search method? Recent research (Luke, 2001; Loveard, 2003) has treated the algorithm similarly to single solution hill-climbing method, where optimal population sizes are combined with run length parameters to search for the best arrangement of computation in a multi-run genetic programming algorithm. While these papers suggest more robust methods, they present genetic programming behaviour as a more local search method, or a hill-climber.

Much of the space of genetic structures has been shown to be difficult to reach by canonical genetic programming (Daida, 2002; Daida et al., 2003b; Langdon, 2000b). If good solutions are in these spaces, it will be difficult to discover them. Previous research also emphasized the shape and content convergence in genetic programming (McPhee and Hopper, 1999; Igel and Chellapilla, 1999), where tree shapes and their contents in the population became increasingly fixed from the top down early in the evolutionary process.

If the role of the population is to provide diversity while hill-climbing toward a single solution, then directly modelling genetic programming as a hill-climber would allow smaller population sizes in combination with elitism and diversity to hill-climb effectively. If the role of the population is to carry out a more global and parallel search over the search space, perhaps more careful selection, recombination and diversity strategies are required.

3.3.2 Representation and Operator Conflicts

Section 3.2 described issues in the representation and operator that increase problem difficulty. Intuitively, exchanging subtrees between two syntax trees (representing computer programs) 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 (Angeline, 1997; Luke and Spector, 1998; O'Reilly and Oppacher, 1996). However, because the population only represents a portion of the search space at one time, could the population make the search more robust against landscape roughness issues by avoiding particular regions of the search space? If the population can encourage the search toward regions of the search space that minimise internal conflicts between the representation and operator, it would be able to 'smooth' the search landscape.

Homology attempts to address some of the problems with the representation and operator. Intuitively, recombination should modify similar content with similar context in parents, thereby preserving the semantics of exchanged genetic material. In this way, solutions may be iteratively and incrementally built. Homology can be achieved by using operators that search for semantically similar subtrees. Platel et al. (2003) defined a homologous crossover for a linear genetic programming representation. Their results showed that homology reduced code growth and produced far more neutral events than a standard crossover operator. While neither method produced many "Advantageous" events, the number of destructive events were more numerous using standard crossover. Smaller solution sizes with mixed improvement of quality were seen in other studies using homologous operators (D'haeseleer, 1994; Langdon, 2000b; Poli and Langdon, 1998a; Nordin et al., 1999).

The concept of homology could also be implemented with a population that consist of similar trees. In this case, operators are more likely to pick subtrees from semantically similar areas as probability of selection will apply similarly to the homogeneous population. That is, less genetically diverse, or homologous, populations may, among other things, improve the chance of preserving semantics during recombination.

As noted in (Langdon, 2000b), a potential drawback of homologous crossover is the slow growth in size of the population. If the initial population is of the wrong structure, e.g. trees much smaller than optimal size, then homologous crossover will approach this rate slowly. Standard crossover might better obtain the ideal size, but its ability to improve solution quality or maintain the size is debatable and problem dependent. Additionally, homologous operators introduce or preserve similarities in the population, which could counter the possible benefits of diversity preservation. However, a converged population, while possibly improving search, can also limit the ability to avoid local optima.

3.4 Summary

This chapter stated three questions that are used to guide the following research. These questions highlight the difficulty in measuring and controlling diversity, the importance of diversity on other aspects of search and the need to understand how and to what extent diversity guides search.

Genetic programming is an appealing search heuristic because it provides an intuitive way to search a very complex space. This representation provides many possible measures and methods of diversity. It is not clear what type of diversity is more relevant for improving search, or what level of diversity would be best. Next, Chapter 4 looks at diversity measures and the correlation between fitness and diversity. Chapter 5 then uses a novel diversity measure to explore the effects of increased diversity, the expected diversity loss and the type of search genetic programming appears to be carrying out.

Issues such as code growth, bloat and problem difficulty and homology are not the primary focus of this thesis. Instead, Chapter 6 examines population diversity to understand its role and the effects it has on the previous issues. As the population is connected to most issues in search, it is likely that many interesting relationships exist.

The population and operator are responsible for producing new individuals during search. While diversity is obviously beneficial to sampling new individuals, do the most dissimilar individuals produce good individuals? Should the search method focus explicitly on dissimilar individuals or simply use them to provide variety for a converging population? These issues are fundamental to how genetic programming searches and also to developing new methods that can be used to improve search. Chapter 7 explores the concept of the survivability of dissimilar and fit individuals to better understand how these individuals guide search.

CHAPTER 4

ANALYSIS OF DIVERSITY MEASURES

Previous investigations into measures of diversity have given the community a clearer view of populations and the search process of genetic programming (McPhee and Hopper, 1999; Darwen and Yao, 2001; Ekárt and Németh, 2002; Rosca, 1995a; Keijzer, 1996; Langdon and Poli, 2002). However, the many different possible definitions of diversity can be conflicting. Identifying the measures that correlate with run performance will enable the design of more efficient operators and genetic programming algorithms.

The three main issues addressed in this chapter are:

1. The ways of measuring and controlling diversity,
2. The correlation between the best fitness and diversity of populations, and
3. The importance of diversity at different stages of the evolutionary process.

As genetic programming is a stochastic algorithm, clear (and always applicable) rules about exact ideal levels of diversity are not expected to be found. The goal is instead to draw general conclusions and 'rules of thumb' about expected diversity and search performance. A survey of diversity measures and diversity preserving methods is presented next. This is followed by an experimental study of the trends of various measures, the correlation between fitness and diversity measures and a discussion about the overall effects of diversity.

4.1 Diversity Measures

Biological diversity refers to the differences between individuals in a population, which in nature imply structure and behaviour difference. Koza (1992) used the term *variety* to indicate the number

of different genotypes in a population. In a standard genetic programming population, this would be the number of structurally unique individuals. While this measure is probably the least informative it is the most common due to its ease of use and understanding. Langdon (1998a) argued that genotypic diversity is a sufficient upper bound of population diversity. Due to the nature of most genetic programming systems and problems, two identical genotypes will produce the same fitness. Thus, a decrease in genotype diversity (unique individuals) will necessarily cause a decrease in phenotype diversity (unique fitness values). In the treatment of the *stack* problem, Langdon (1998a) investigated the effects of the crossover operator on variety. The author noted that genetic programming loses some ability to improve fitness after 20-30 generations, which is most probably due to selection and crossover causing a loss of variety. Langdon also noted that in the *stack* problem, in runs with better fitness, crossover appeared to produce a larger number of fitter (and non-duplicate) children than their parents. Also, the author found that the loss of diversity caused a decrease of unique terminals that, due to subtree crossover, led to further diversity loss. Langdon and Poli (2002) later noted that measuring variety with only unique genotypes fails to consider the ancestral history of individuals, the degree of difference between non-unique individuals and their behavioural similarities.

The standard representation in genetic programming (syntax trees) lends itself to more fine grain structural measures that consider nodes, subtrees, and other graph theoretic properties (rather than just entire trees). Keijzer (1996) measured subtree variety as the ratio of unique subtrees over total subtrees, and program variety as the ratio of the number of unique individuals over the size of the population. Keijzer also used a distance measure between two individuals as the number of distinct subtrees the individuals share. The author noted that his distance measure of distinct subtrees between two individuals could be used to predict when subtree crossover will fail to provide improvements due to loss of distinct subtrees and diversity. Tackett (1994) also measured structural diversity using subtrees and schemata frequencies.

Problem specific measures can allow additional insight into population diversity, especially on novel and non-traditional problems. D'haeseleer and Bluming (1994) defined *behaviour* and *frequency* signatures for each individual based on fitness and *gene* (i.e. specific nodes) frequencies, respectively. The average correlation between the respective signatures in the population represents the *phenotype* and *genotype* diversity of the population. In addition, D'haeseleer and Bluming tagged genetic code and evaluated the behaviour of individuals with a "stimulus-response map" to gain further knowledge into the structure and behaviour of populations in their robot tank problem. Using these measures, the authors witnessed emerging demes (distinct subpopulations) with neighborhood selection and mating.

The degree of graph isomorphism could be applied to genetic programming trees as a measure of diversity. However, the properties of functions used in genetic programming (associativity, com-

mutativity, etc) would require special, and possibly complex, implementations of isomorphism (Rosca, 1995a). Also, determining graph isomorphism would be computationally expensive for an entire population. However, a measure of possible isomorphic trees could be found by noting simple properties (terminal, functions, depth, etc.) to determine the individuals which could be isomorphic without actually computing isomorphism.

McPhee and Hopper (1999) investigated diversity at the genetic level by assigning numerical tags to each node in the population. The tags track the survival of nodes from the initial population and the change of context for nodes during recombination. The authors also tracked the genetic lineages from the initial population by noting the individuals selected for recombination, which child they produced and which of the parents provided the root portion of the child's tree. They found that populations in the final generation often descended from one single initial individual, since genetic lineages were often reduced to one surviving lineage early on in the evolutionary process.

Measuring the difference between two individuals based on string edit distances has been used several times in genetic programming. O'Reilly (1997) used an edit distance based on string matching, which uses single node insertions, deletions and substitutions to transform two trees to be equal in structure and content. De Jong et al. (2001) also used a similar edit distance in a multiobjective method. This edit distance, a modification of string edit distance, or Levenshtein distance, overlaps two trees at the root node. Two different nodes, when overlapping, score a distance of 1. Equal nodes score a distance of 0. The edit distance is then the sum of all different nodes, which can be normalised by dividing it by the size of the smaller tree. The measure represents the number of node changes that need to be made to either tree to make them equal in structure and content. Note that this measure does not take into consideration the standard subtree crossover operator, but does resemble a single point mutation operator.

O'Reilly (1997) noted the importance of using structural distance measures on genetic programming populations to understand the underlying dynamics. An edit distance measure was used here to study the effects of crossover and the differences between individuals and better individuals. While no clear results were found, the ability to understand genetic programming populations with edit distance measures was suggested.

Ekárt and Németh (2000) defined an edit distance specific to genetic programming syntax trees, adapted from (Nienhuys-Cheng, 1997), which considered the cost of substituting between different node types (functions vs. terminals and within these classes). The difference between two trees is defined as:

$$dist(T_1, T_2) = \begin{cases} d(p,q) & \text{if neither } T_1 \text{ nor } T_2 \text{ have any children} \\ d(p,q) + \mathrm{K} * \sum_{l=1}^{m} dist(s_l, t_l) & \text{otherwise} \end{cases}$$

where T_1, T_2 are trees with roots p, q and possible children (m total) subtrees s and t. The overall distance between two trees, $dist(T_1, T_2)$, is found by recursively finding the distance between their nodes, $d(p, q)$. The constant K is set to $\frac{1}{2}$ but can be adjusted to weight the depth of tree differences differently. Two trees are brought to the same tree structure by adding "null" nodes to each tree. Note that the differences near the root have more weight. This is a convenient description for genetic programming as it has been noted that programs converge quickly to a fixed root portion (Igel and Chellapilla, 1999; McPhee and Hopper, 1999). As before, this measure does not account for the subtree crossover operator, but it does represent its dynamics more closely by giving a higher cost to the upper portions of trees which are more difficult to change when using subtree crossover. Additionally, the cost of changing one node to another can be specified for each pair of nodes or for classes of nodes, as was the case in (Ekárt and Németh, 2000).

Measures based on behaviour compare differences among the populations' fitness values at a given time. Rosca (1995a) used the fitness values in a population to define an entropy and free energy measure. Entropy represents the amount of disorder of the population, where an increase in entropy represents an increase in diversity. Rosca found that populations appeared to be stuck in local optima when entropy did not change or decreased monotonically in successive generations.

Entropy is calculated for a population by first placing fitness values into classes and noting the size of each class (Rosca, 1995a). In a discrete fitness space, a unique fitness value may define a fitness class. In continuous spaces, discretisation may be necessary. The level of convergence in the genotype space often significantly reduces the number of unique fitness values, even in continuous spaces. Given a fitness value i, k unique fitness values in the current population, and fitness classes $p_1 \dots p_k$, the fitness class p_i is the proportion of the population which has fitness equal to i. Entropy is then defined as,

$$-\sum_k p_k \cdot \log p_k.$$

In physical systems, entropy represents the amount of chaos in the system. In genetic programming, high entropy reveals the presence of many unique fitness values, where the population is evenly distributed over those values. Low entropy describes a population which contains fewer unique fitness values as many individuals have the same fitness, as shown in Figure 4.1. Entropy gives insights into the distribution of fitness values over the population, but does not describe the precise distribution of the fitness values. Additionally, because entropy describes the number of unique fitness values and some information about their distribution, it also describes the same information used by selection. If all the individuals in the population have the same fitness value, selection based on fitness becomes random. The higher the entropy, the more evenly distributed the population becomes over the possible fitness values, the more fine-grain the population will appear to selection. Thus, entropy can describe, as does phenotype diversity in a broader sense, the dynamic change of the applied selection pressure.

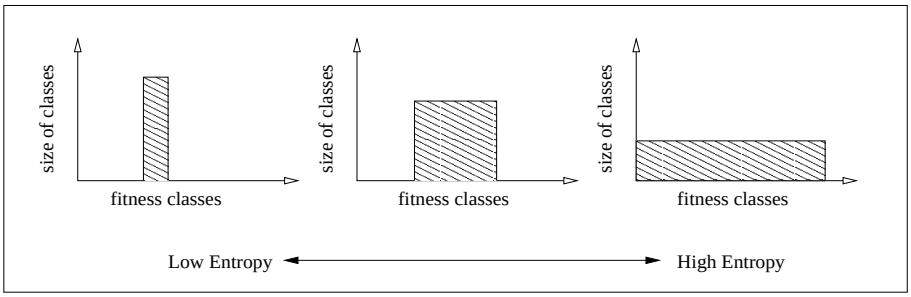


FIGURE 4.1: Examples of entropy distributions from low entropy, where all members of the population have the same value, to high entropy, where each individual is represented by a different value (assuming there are at least as many possible fitness values as individuals).

The frequency of the terminals and functions in a population could also serve as a measure of population diversity. Similarly, diversity could also refer to the number and type of tree structures that the population represents. A visualisation of a genetic programming tree was described for a circular lattice in (Daida, 2002) and later in (Daida et al., 2003a), where a population visualisation using colors represented the most sampled type of shapes in the lattice. Using the lattice model, a population visualisation method can be developed using a 3-dimensional graph (Gustafson and Krasnogor, 2003), which gives rise to measures that describe the tree shapes sampled by the population. Figure 4.2 shows a full binary tree of depth 10 on the circular lattice in the upper left graph. A 3-dimensional visualisation can be made by finding the cumulative node count of the population for the absolute node positions represented in the lattice. This cumulative count then can denote the height of each node, as shown in the bottom two figures of Figure 4.2, where the upper right graph shows the same visualisation from above.

4.1.1 Promoting Diversity

The canonical view of evolution and diversity is that more diversity will provide more opportunities for evolution. However, typical evolutionary algorithms contain a phase of exploration followed by exploitation (Eiben and Schippers, 1998). Promoting all kinds of diversity during the entire search process could be counter-productive to the exploitation phase. The type and amount of diversity required at different times remains unclear. However, several measures and methods have been used to promote diversity. These methods typically use a non-standard selection, mating, or replacement strategy to increase or control diversity. Common methods include geographical distributions of individuals that limit their interactions, such as neighbourhoods (Collins, 1992) and islands (Martin et al., 2000). Other methods consider behaviour and structure similarities, such as sharing (Goldberg and Richardson, 1987), crowding (DeJong, 1975) and genotype sharing (Deb and Goldberg, 1989). These techniques were initially applied in genetic algorithms.

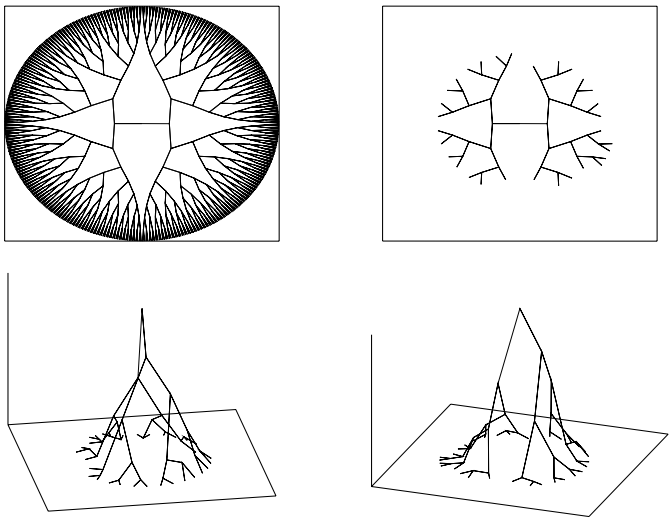


FIGURE 4.2: Example of population visualisations on a circular lattice in the upper left. A sample tree is shown in the upper right, which actually represents an overhead view of a population of trees where the node height is the number of times the population samples a node at this structural location. The 3-dimensional views of this population are in the bottom two graphs.

Eshelman and Schaffer (1993) investigated the advantage of pair-wise mating in genetic algorithms. The authors used Hamming distances to select individuals for recombination and replacement, finding improvement over hill-climbing-type selection strategies for genetic algorithms. Ryan's "Pygmie" algorithm (1994) addressed premature convergence and elitism in small populations for evolving minimal sorting networks. To facilitate selection for reproduction, the algorithm builds two lists: one based on fitness and the other on length. Ryan's algorithm maintained higher diversity and prevented premature convergence using simple measures. De Jong et al. (2001) used multiobjective optimisation for the n-Parity problem to promote diversity and concentrate on non-dominated individuals, according to a 3-tuple of <*fitness, size, diversity*>. Diversity was the average square distance to other members of the population, using a specialised measure of edit distance between nodes. This multiobjective method promoted smaller and more diverse trees.

Keller and Banzhaf (1995) described a structural difference measure based on edit distance between genotypes. The measure was more complicated than standard edit distance, but was intended for explicitly controlling the diversity of populations. Brameier and Banzhaf (2002) used a string edit distance on the effective portions of their *Linear Genetic Programming* individuals, measuring the distance between program code that contributed toward fitness. They used their measure in a two-level tournament, selecting for fitness and then for diversity.

McKay (2000) applied the fitness sharing concept from Deb and Goldberg (1989) to test its feasibility in genetic programming. Fitness sharing was credited with maintaining population diversity, allowing performance improvement and population size reduction for the Multiplexer and recur-

sive list membership problems. Diversity was the number of fitness cases solved, where the sharing concept assigned a fitness based on an individual's performance divided by the number of other individuals with the same performance. Also, McKay studied negative correlation from (Liu et al., 2000) and a *root Quartic negative correlation* (2001a,2001b) to preserve diversity on the Multiplexer problem with mixed results. Similarly, a selection method that is uniform over the fitness values was suggested as an alternative way to preserve diversity (Hutter, 2002). Ekárt and Németh (2000) applied fitness sharing with a novel tree distance definition to a symbolic regression problem, suggesting that it may be an efficient measure of structural diversity. Their results showed promise for controlling the size of programs, although not initially improving performance. The authors then applied their measure between every pair of individuals in a weighted arithmetic mean to develop a population diversity measure (Ekárt and Németh, 2002). This measure was used to adaptively control diversity for explorative and exploitative search phases, noting the conflict between fitness improvement and high diversity observed in previous work. The authors found that on their symbolic regression instances, fitness sharing was able to improve accuracy and maintain population diversity.

Bersano-Begey (1997) tracked the number of individuals that solved specific fitness cases. The method promoted diversity and the discovery of different or less popular solutions. This was similar to the *Stepwise Adaptation of Weights* technique for constraint satisfaction and symbolic regression instances (Eiben and van Hemert, 1999; Eggermont and van Hemert, 2001). Smith et al. (1993) investigated diversity within their immune system algorithm for classifier systems, based on a standard genetic algorithm. Their task was not concerned with *traditional* optimisation and required diverse populations to be successful. A speciation tree using Euclidean distance was applied by Bessaou et al. (2000) in a study on multimodal optimisation with island models. Their algorithm divided individuals into species, evolved them with a genetic algorithm and then redistributed them into new species. Geard and Wiles (2002) counted unique genotypes while studying recombination and diversity for a genetic algorithm solving the "royal staircase" problem.

Fernandes and Rosa (2001) looked at varying population sizes and non-random mating to maintain diversity for the Royal Road problem. Their negative assortative mating looks for genotypes with maximal Hamming distances. Darwen and Yao (2001) studied cooperation in the Iterated Prisoner's Dilemma problem and found that increasing behavioural diversity, not genetic diversity, can improve cooperation and performance. The authors also commented on the dogma surrounding diversity and some previous methods to maintain diversity (Darwen and Yao, 2000). Ursem (2002) cited the importance of high and low diversity phases in an evolutionary strategy framework. The author used a "distance-to-average-point" diversity measure for the real-value encoded individuals. Depending on whether the diversity is in a predefined high or low diversity phase, different recombination operators were used, allowing diversity to fall or promoting more diversity.

TABLE 4.1: Experiment and problem parameters for the Ant, Parity, Quartic and Rastrigin diversity measure experiments. These parameters settings represent the commonly used settings in the literature for these problems. The division and log functions are protected, as described in Chapter 2.

Evolutionary algorithm	Generational
Population size	500
Stopping criterion	Maximum generation = 51
Function sets	
Ant	if_food_ahead, progn2, progn3
Parity	and, or, nand, nor
Quartic, Rastrigin	+ , -, *, p/, sin, cos, exp, log
Terminal sets	
Ant	left, right, move
Parity	D1, D2, D3, D4, D5
Quartic, Rastrigin	x
Tree generation	Ramped half-n-half
Initial depth	4
Maximum depth	10
Subtree crossover probability	1.0
Subtree crossover internal node selection prob.	0.9
Mutation probability	0.0
Selection method, size	Tournament selection, size 4

4.2 Empirical Analysis of Diversity Measures

The Artificial Ant, Even-5-Parity, and symbolic regression problems (using the Quartic polynomial and Rastrigin function) were introduced in Chapter 2. The problems are used here to explore diversity measures and the relationship between diversity and fitness. Table 4.1 contains the problem and experiment parameters for the following empirical investigation. These parameters were selected as they are commonly used in the literature and in many similar empirical studies. Note that the function set used here is typical for the Rastrigin function, whereas the Quartic problem typically only uses addition, subtraction, multiplication and division. The same function set are used for both to be consistent and do not use any ephemeral random constants. The Rastrigin problem is likely to be more difficult to solve without using ephemeral random constants. All experiments in this thesis were carried out using a modified version of the ECJ framework (Luke, 2004) and the Mersenne Twister random number generator (Matsumoto and Nishimura, 1998).

4.2.1 Diversity Measures Used

The following measures are collected for each population in every generation, whereby the diversity present in a population can then be compared to other populations.

Genotype Diversity

Genotype diversity represents the number of unique trees (Langdon, 1998a). Genotype diversity does not consider the fitness or behaviour of the trees. Two trees are equal only if they contain the exact same structure and content.

Phenotype Diversity

Phenotype diversity represents the number of unique fitness values in a population. The number of possible fitness values is determined by the fitness function and varies between problem domains. For example, in the Parity problem, there is a finite number of possible fitness values that an individual can have. Instead, the fitness space is continuous in regression problems, but due to the precision of numbers, wrappers around operators (protected division for instance) and the presence of non-functional code it is possible for different trees to have the same fitness.

Entropy

Entropy is calculated as described in Section 4.1 and in (Rosca, 1995a). A fitness class represents each unique fitness value in the current population.

Pseudo-Isomorphic Tree Diversity

In an attempt to approximate the degree of isomorphism, we count the number of pseudo-isomorphic trees in the population. Pseudo-isomorphic trees are represented by the number of similar 3-tuples in the population, where a 3-tuple is defined as <# *of terminals, # of nonterminals, depth*> for each individual. Two identical 3-tuples represent trees which could be isomorphic, while two non-identical 3-tuples represent trees that certainly can not be isomorphic.

Edit Distance One (non-weighted)

Edit distance One diversity is based on the standard edit distance, described in Section 4.1 and in (de Jong et al., 2001; O'Reilly, 1997), and is referred to as "ed 1" in most figures. Trees are brought to the same structure and overlapped. The number of non-identical overlapping nodes between the two trees are counted. This is then normalised by tree size.

Edit Distance Two (weighted)

Edit distance Two diversity is based on the depth-weighted edit distance between individuals used by Ekárt and Németh (2000). This measure (denoted "ed 2") is adapted back to its original formulation (Nienhuys-Cheng, 1997) where the difference between any two non-equal nodes was 1. Using a value of $k = \frac{1}{2}$ gives differences near the root more weight.

In (Wineberg and Oppacher, 2003) an $O(n + m)$ inter-population diversity method is developed based on pair-wise distance by counting the frequencies of symbols for each position in the genome. While a similar method could be found for genetic programming syntax trees, the variable length and size of symbol sets would make this calculation more complex. To reduce computation time here, an approximate population diversity measure is found by only comparing each population member against a single tree. Every individual in the population is compared with the best fit individual found so far in the run. This measure is then divided by the population size.

Both edit distance One and Two are vulnerable to outliers, especially when the best fit individual is the outlier. However, previous experimental results display two key properties which make these measures appropriate and representative. First, even if the best fitness is found in the initial population, an individual in the current generation is considered to be the best of the run if it is at least as good as the current best of the run individual. Secondly, with probabilistic selection based on fitness, the best individual is likely to contribute several offspring to the next generation and is unlikely to remain the outlier for long. Later chapters also consider the best individual in the current generation for these measures. Another reason for using the best of run individual in this chapter is that it is common for researchers to consider this individual during analysis rather than the best in the current generation.

4.2.2 Correlation Measures

An objective of this study is to quantify the importance and ideal levels of diversity, recorded by different measures on common problems. The primary test of the relationship between diversity and fitness will be the Spearman correlation measure (Siegel, 1956). The Spearman measure ranks two sets of variables and tests for a linear relationship between the variables' ranks. Correlation is first examined to determine if two runs can be distinguished by their diversity in terms of which run is better. As interesting relationships could easily exist but may not necessarily be linear, a range of scatter plots of diversity measures and fitness are evaluated.

The Spearman correlation coefficient is computed as follows:

$$1 - \frac{6 \sum_{i=1}^{N} d_i^2}{N^3 - N},$$

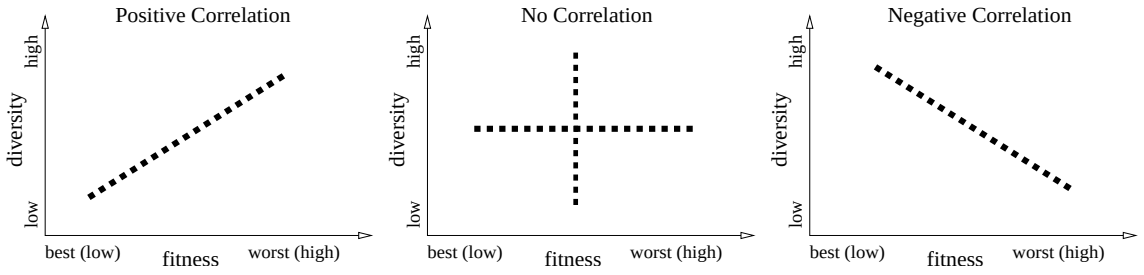


FIGURE 4.3: Examples of ranked correlation scatter plots between fitness and diversity, where low fitness (ideal) is ranked from 1 to 50 (with 50 runs total) and diversity is ranked from high (1) to low (50). The middle graph shows the case of no correlation where the points are aligned vertically *or* horizontally.

where N is the number of runs and d_i is the distance between each population's fitness rank and diversity rank. A value of -1.0 represents negative correlation, 0.0 denotes no correlation and 1.0 demonstrates positive correlation. For the measures used here, when low best-of-generation fitness values, which will be ranked in ascending order (1=best,...,50=worst), occur with high diversity, ranked in ascending order (1=lowest diversity and 50=highest diversity), the correlation coefficient should be strongly negative. Alternatively, a positive correlation indicates that either bad fitness accompanies high diversity or good fitness accompanies low diversity. Figure 4.3 shows the relationship between fitness on the X-Axis and diversity on the Y-Axis and the type of correlation that a scatter plot in these circumstances would indicate.

4.3 Analysis of Results

Fifty independently random runs, with one graph for each problem and diversity measure are shown in Figures 4.4 to 4.12. Figure 4.4 shows the best fitness of each generation during the evolutionary process and Figure 4.5 shows the evolution of size, while depth is shown in Figure 4.6. Many runs stop improving after 15-20 generations, with the exception of the Parity problem which continues to make improvements. Previous research by Luke (2001) showed that it is better to carry out short runs (above a *critical point*) than fewer long runs for the Ant and Quartic problem. Luke also found that with the Parity problem (Even-10), one long run was actually better. This was due to the difficulty of the problem and the ability of genetic programming to consistently make improvements. This *critical point* was around generation 8 for the Quartic problem and slightly higher for the Ant problem. An early period of higher activity in the runs also exists with respect to diversity measures. There is typically a lot of activity in the early generations and not too much after generation 30.

For the Quartic and Rastrigin experiments, the phenotype diversity in Figures 4.7 shows an ini-

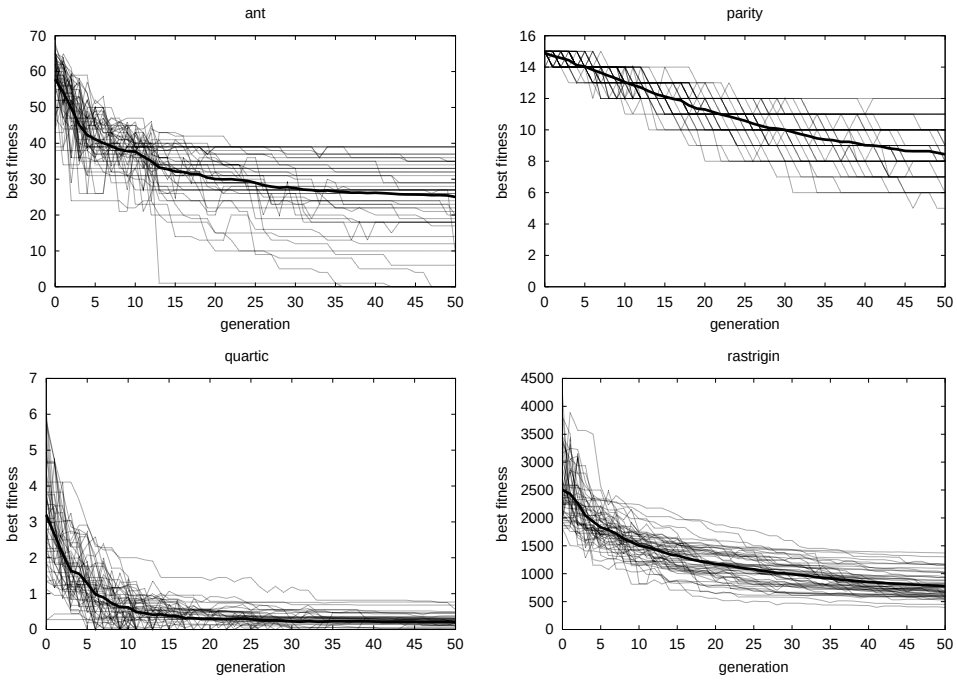


FIGURE 4.4: Ant, Parity, Quartic and Rastrigin best fitness in population, plotted against the generation number. 50 independently random runs of each problem are shown.

tial decrease followed by a sharp increase. This behaviour was also seen with genotype diversity and entropy: an initial sharp decrease was followed by an increase within the Quartic and Rastrigin experiments and in all experiments with genotype diversity. Intuitively, the cause of this initial fluctuation is due to the ease with which improvements can be found in the initial solutions. This initial phase highlights the differences between the experiments. Also, note that phenotype diversity for the Parity experiments continues to increase until the final generation.

For all experiments, the edit distance One in Figure 4.9 generally decreases after the initial generation. Also, in Figure 4.10, the populations measured with edit distance Two behave similarly[1]. With this in mind, and because the edit distance Two measure places more importance on the root and higher portions of trees, one can conclude the following: While trees are changing (according to edit distance One) to be more like the best fit tree in each population, the differences between

[1] Figure 4.10, plotting edit distance Two diversity, shows an initial increase followed by a decrease in most experiments. The decrease comes at a later stage than that seen with edit distance One. While this might seem to imply that their is an increase in diversity with respect to root portions of the trees, which occurs for a longer amount of time, two other issues need to be considered. First, the K value is 0.5, meaning that each depth level in the tree (for binary trees) is capable of contributing the same to the distance measure. Secondly, the Ant problem uses a 3-arity function and the regression problems both contain unary functions. These two issues explain why a more dramatic increase in edit distance Two diversity is seen in the Ant problem (due to the 3-arity function) and less in the regression problems (due to the unary function) and an overall later decrease in all problems (due to the trees still growing toward maximum depths in combination with the K value, as seen in Figure 4.6). The performance of both measures, edit distance One and Two, can also be seen for the same problem domains in Chapter 5, Figure 5.2. In this case, it is more clear what increased diversity under the edit distance Two measure looks like.

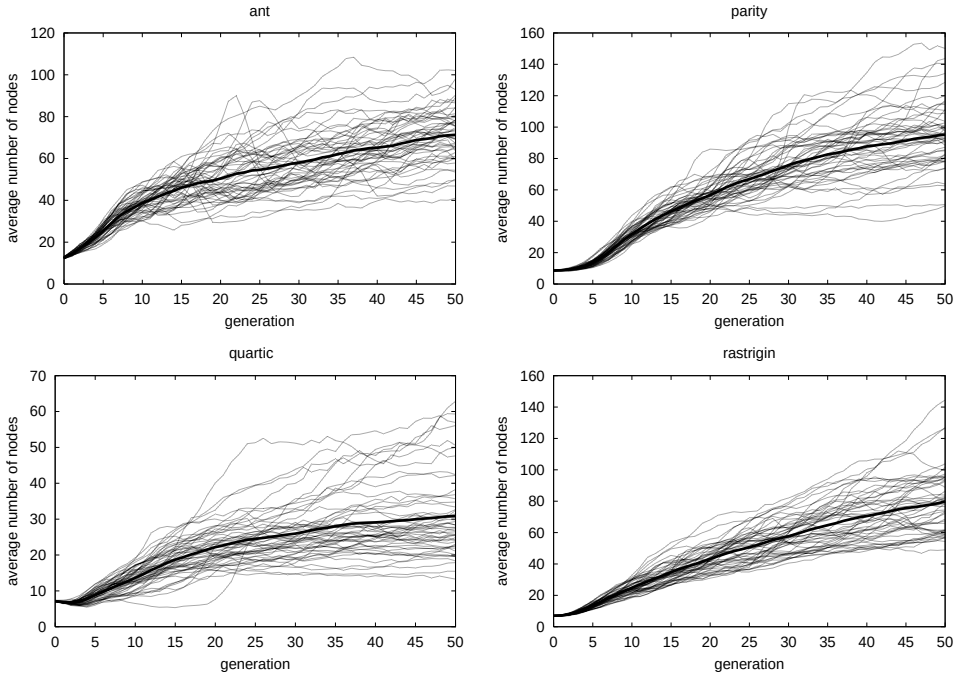


FIGURE 4.5: Average number of nodes vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

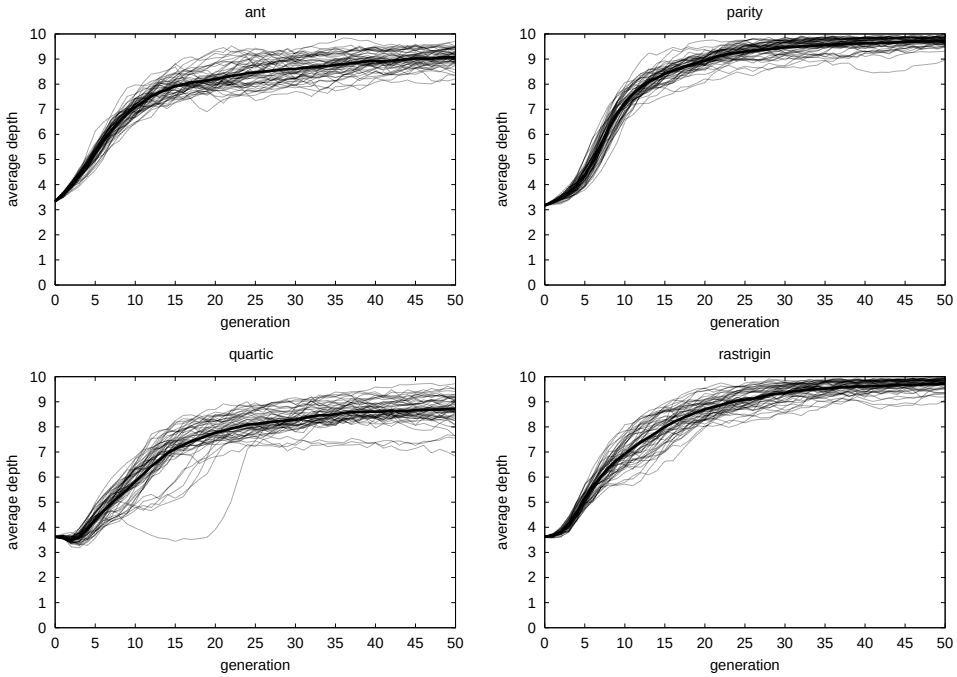


FIGURE 4.6: Average depth vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

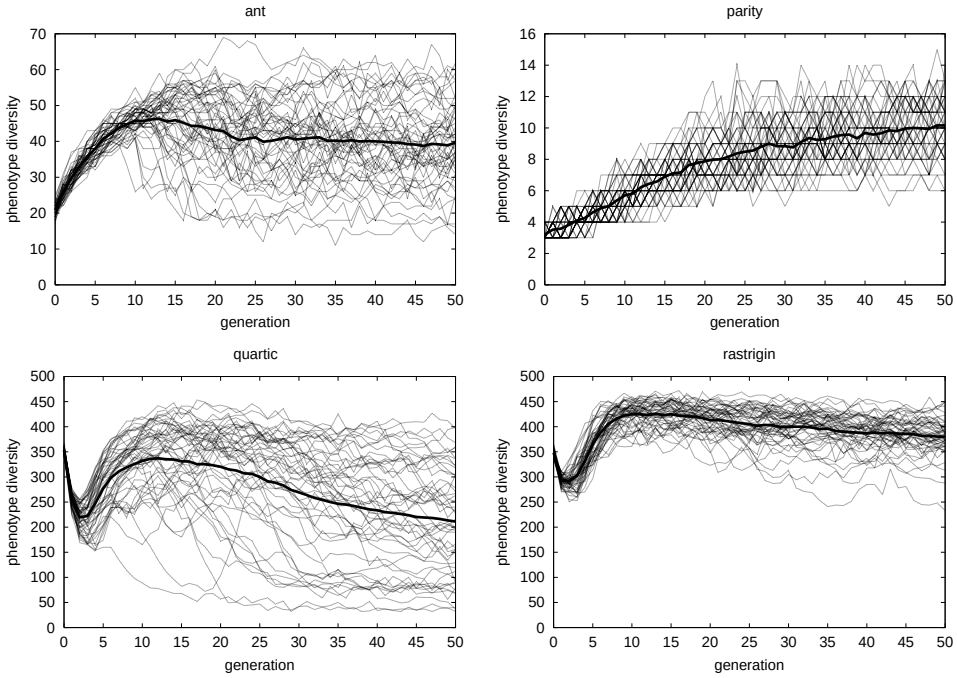


FIGURE 4.7: Ant, Parity, Quartic and Rastrigin phenotype diversity, plotted against the generation number.

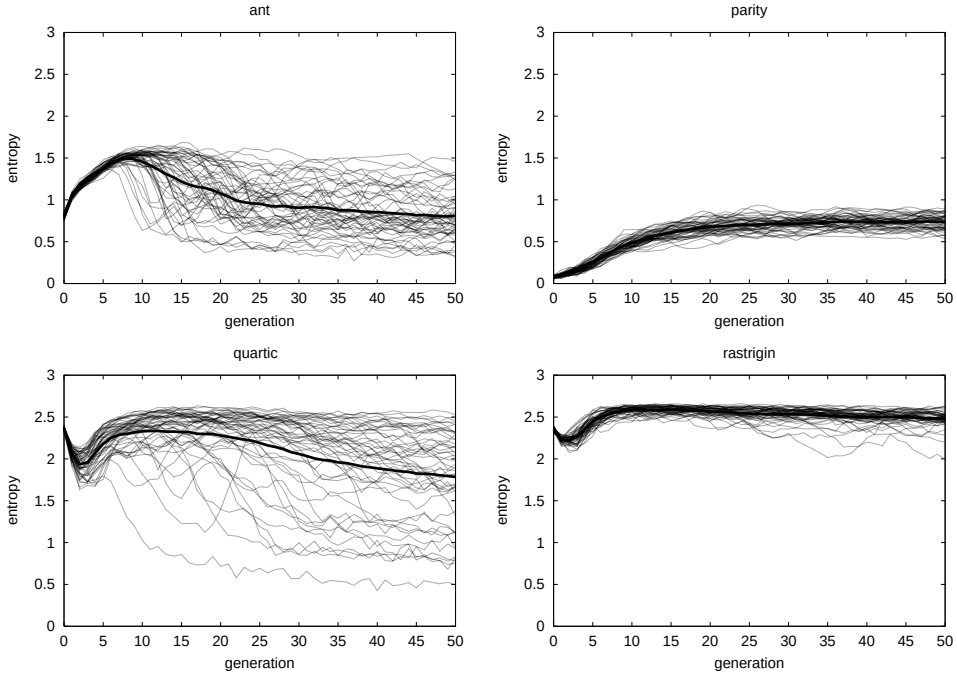


FIGURE 4.8: Average entropy vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

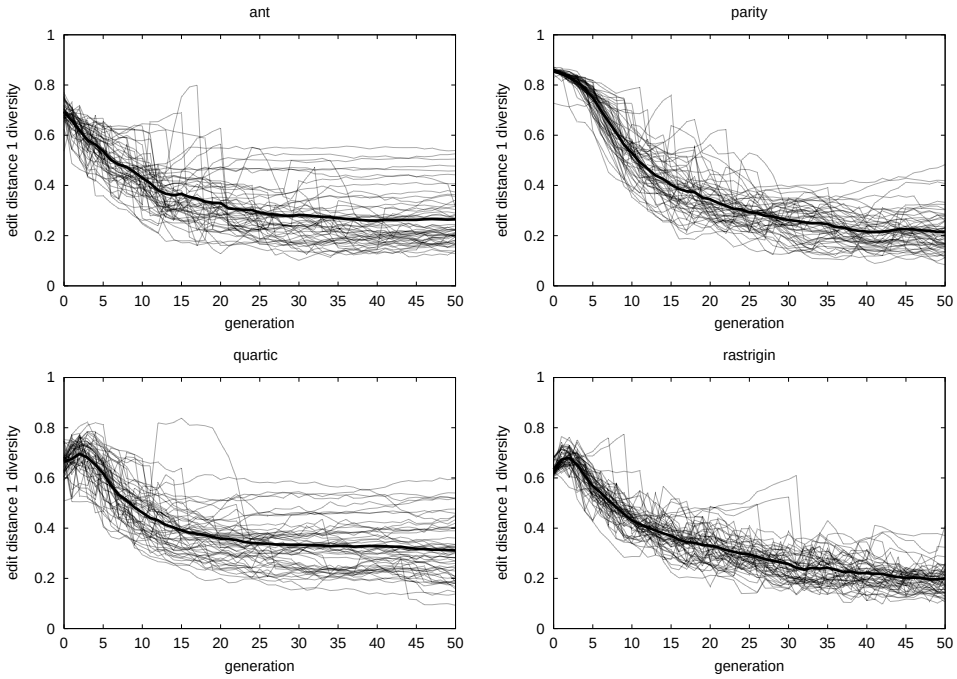


FIGURE 4.9: Ant, Parity, Quartic and Rastrigin edit distance One diversity plotted against the generation number.

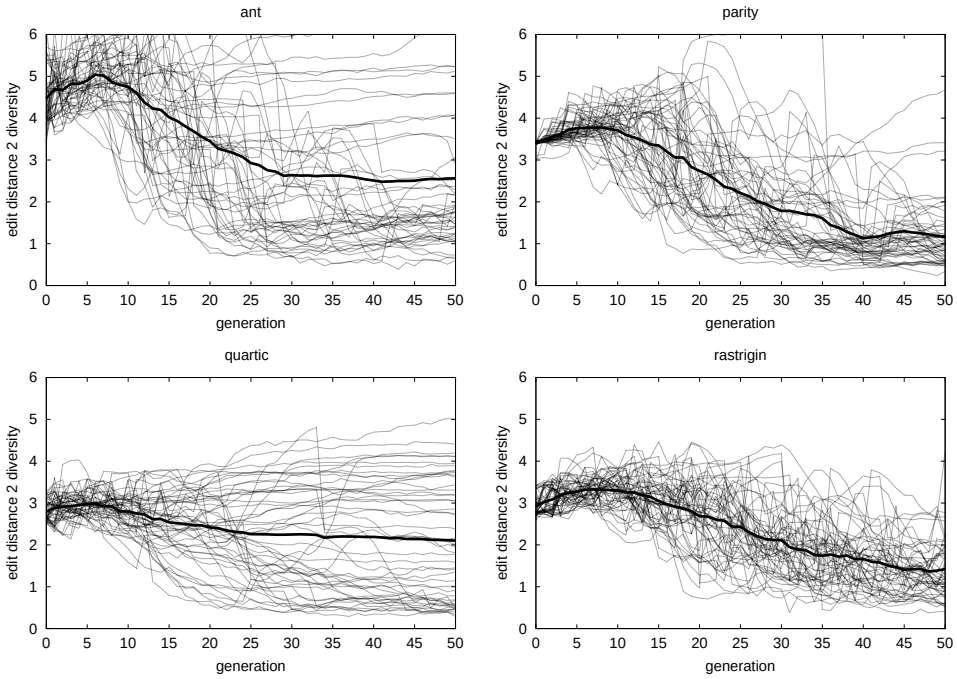


FIGURE 4.10: Edit distance Two diversity vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

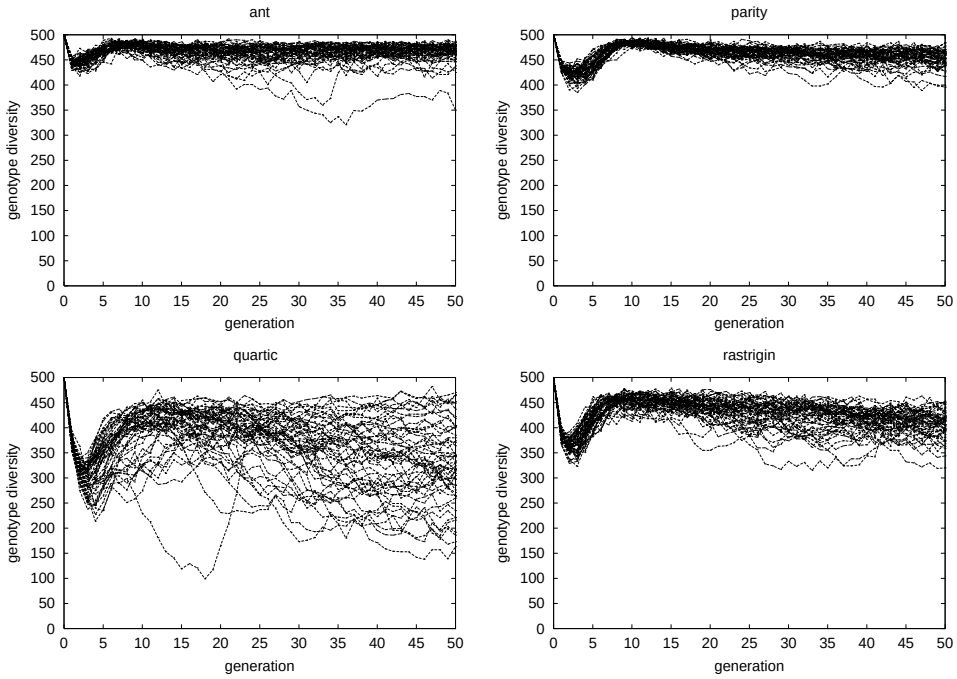


FIGURE 4.11: Genotype diversity vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

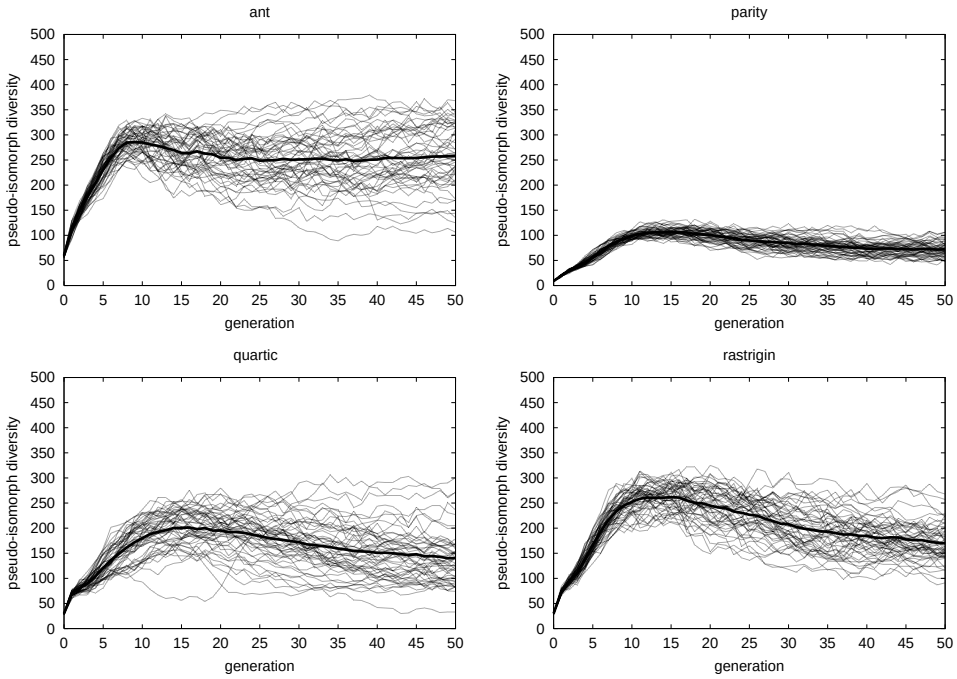


FIGURE 4.12: Pseudo-isomorph diversity vs. generation for the Ant, Parity, Quartic and Rastrigin experiments.

the roots and top portions of the tree also become more similar (according to the edit distance Two measure). This supports previous conclusions (Igel and Chellapilla, 1999; McPhee and Hopper, 1999; Soule and Foster, 1998) that roots become fixed early on in the evolutionary process. Structural convergence is important when considering using a method to control diversity. If structural convergence is beneficial to genetic programming search, then encouraging or forcing structural diversity (edit distance in this case) could have negative consequences. However, the loss of edit distance diversity does not necessarily mean a loss of phenotype diversity or the worsening of fitness, as seen in Figure 4.7 and 4.4.

Lastly, the figures show the behaviour that in some runs fitness continues to increase until the final generation. Identifying the dynamics and properties that allowed for this continued increase is critical for genetic programming practitioners. This is a goal of this research: understanding how to make populations more amenable to improvement. Given the wide range of fitness and diversity, one would like to know if these measures correlate with fitness in any way. Addressing this question is key to understanding if controlling diversity is likely to be effective and how it should be applied to different problem domains.

4.3.1 Correlations in Final Populations

Table 4.2 summarises the Spearman correlation coefficients between the best fitnesses in 100 runs and the diversity in the final populations. Also the correlation between diversity measures is reported.

In the Ant experiments, negative correlation is seen between phenotypes and fitness and also between entropy and fitness. Good (low) fitness is seen with high phenotype diversity and entropy. There is a positive correlation of edit distance with fitness and also between pseudo-isomorphs and fitness. Only very weak correlation is seen between genotypes and fitness in Ant experiments, which is the trend for all the experiments. In this case, a positive correlation between fitness and edit distance and between fitness and pseudo-isomorphs suggests that low (good) fitness is seen with low diversity. From Figures 4.9 and 4.10, edit distance generally decreases during the run. While runs tend to structurally converge for the Ant experiments, and with respect to the edit distance One measure in the Parity experiments, those which converge more often have better fitness. This may be the result of good fitness being found early in the run, or it may be the result of convergence leading to better fitness.

The table of correlation coefficients in Table 4.2 also gives the Rastrigin experiment results. This table shows the lack of strong correlations between diversity and fitness (the same effect is partially seen in the Quartic experiments as well). It may be the case that a correlation did exist between fitness and diversity, the relationship is not linear or the final populations have lost any correlation

TABLE 4.2: Spearman correlation coefficients for the Ant, Quartic, Rastrigin and Parity experiments.

The Ant Problem

	fitness	phenes	genes	p-isom	entropy	ed 1
phenes	-.3936	–	–	–	–	–
genes	.1962	-.4950	–	–	–	–
p-isom	.4009	-.6389	.6949	–	–	–
entropy	-.3615	.9039	-.5724	-.7569	–	–
ed 1	.4205	-.5040	.2991	.3998	-.4891	–
ed 2	.4606	-.4537	.4702	.5603	-.4949	.7504

The Quartic Problem

	fitness	phenes	genes	p-isom	entropy	ed 1
phenes	.4345	–	–	–	–	–
genes	-.1363	-.0353	–	–	–	–
p-isom	-.0300	.1588	.8408	–	–	–
entropy	.3924	.9730	-.1712	.0070	–	–
ed 1	-.1640	.0045	.2290	.3150	-.0191	–
ed 2	-.0881	-.0273	.1554	.2182	-.0461	.6891

The Rastrigin Problem

	fitness	phenes	genes	p-isom	entropy	ed 1
phenes	-.0616	–	–	–	–	–
genes	-.1305	.7089	–	–	–	–
p-isom	-.2262	.5521	.6163	–	–	–
entropy	-.0402	.9688	.7324	.5525	–	–
ed 1	-.0530	-.0365	.2056	.2014	.0460	–
ed 2	-.0762	.1185	.3265	.3828	.1750	.6514

The Parity Problem

	fitness	phenes	genes	p-isom	entropy	ed 1
phenes	-.7803	–	–	–	–	–
genes	-.0641	.0510	–	–	–	–
p-isom	.0773	.0646	.5132	–	–	–
entropy	-.7146	.7048	-.0379	.0204	–	–
ed 1	.3235	-.2156	.1178	.4483	-.3062	–
ed 2	.0148	-.0087	.2656	.5377	-.0626	.7265

due to the repeated application of selection and recombination without change in fitness.

The importance of phenotype diversity is now seen with the Parity experiments in Table 4.2, where a strong negative correlation exists between fitness and phenotype diversity. Figure 4.7 shows that phenotype diversity tends to increase in the Parity experiments. With only 32 possible fitness values in the problem, the population begins with random guesses near a fitness of 16. As populations undergo selection and recombination, the number of unique fitness values increases from 3-4 to 6-13. Without some increase in phenotype diversity, genetic programming cannot distinguish between good individuals and bad ones.

As tournament selection uses the fitness values of an individual to decide tournaments, fewer unique phenotypes in the population (and the lower the entropy) will make selection more *random*. That is, selection will be faced with many individuals that have the same fitness. Therefore, if high phenotype diversity and entropy are maintained, selection pressure remains at the pre-set level. The lowering of phenotype diversity and entropy might actually benefit some problems where less selection pressure is suitable, but negatively affect others where higher selection pressure is better.

Table 4.2 also gives the correlation between measures of diversity. In the Ant experiments, note that more phenotype diversity negatively correlates with the structural measures (genotypes, pseudo-isomorphs, and the edit distances). An increase (or decrease) of unique fitness values in the population corresponds with a decrease (or increase) in the structural diversity. This seems counter-intuitive as *more* unique genotypes should correspond to *more* unique fitness cases. This behaviour is expected with the edit distance measures as these measures generally decrease during evolution while phenotype diversity increases. In this problem, the discovery of different fitness values appears to be aided by less structural diversity. That is, if the population is structurally similar, it is easier to find more unique fitness values. Possible hypothesese for this behaviour include: a better environment for crossover, less deception in the search space or a more focused local search phase.

4.3.2 Evolving Populations' Correlation

Is diversity more important at different stages of the evolutionary process? The fact that several methods have been previously used to adaptively control the level of diversity would suggest so. Figure 4.13 shows the correlation between diversity and best fitness for each generation. Note that each point represents the correlation between 100 populations, sampled from 100 runs where there is a dependency of later generations on preceding ones.

Both the Ant and Parity experiments contain varying levels of correlation between edit distance and fitness and between phenotype diversity and fitness. The Quartic experiments contains a period of early fluctuation, followed by an increase in positive correlation between entropy (and phenotype diversity) and fitness. As runs typically achieve the best fitness early, this effect may be due to many copies of the best fit individual accumulating in the population. That is, populations which achieve good local optima begin to have lower entropy.

The Rastrigin experiments contain an early period of varying correlation between diversity and fitness, after which most measures lost correlation with fitness. In this problem and representation, the relationship between fitness and diversity becomes less important, probably due to other more critical relationships like node-to-node dependencies (Daida et al., 2001). A positive correlation between fitness and edit distance occurs together with a negative correlation between fitness and

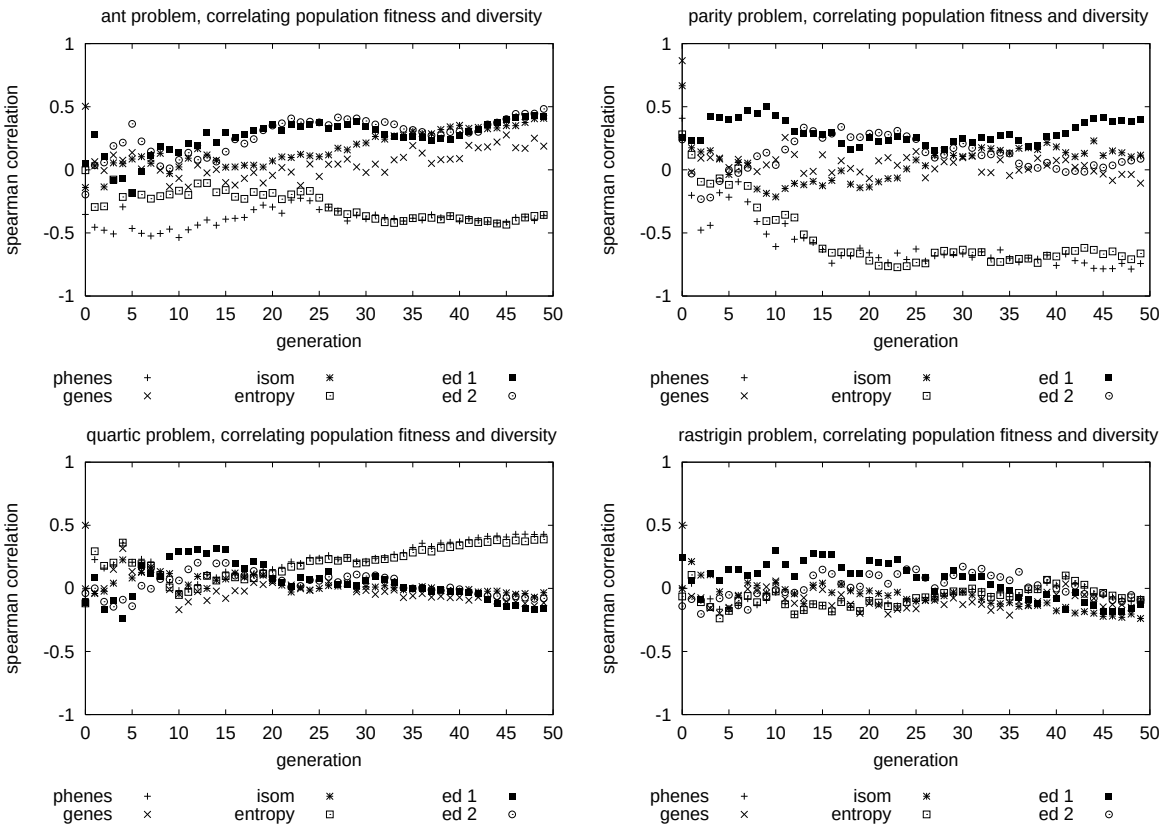


FIGURE 4.13: Evolving populations' correlation between best fitness in each population and different diversity measures. Each point represents the correlation between 100 populations from a 100 runs, each of the 50 generations are represented.

phenotype diversity is seen in the Ant and Parity experiments. These results suggest that the fitness landscape induced by the representation, operator and fitness function is uncorrelated. Small differences between individuals are still capable of expressing a wide range of behaviours. However, this statement should be considered in the light of the operator not being used to define distance and the actual difference between behaviours is not considered. The measures used here only give approximate descriptions of the fitness landscape.

4.3.3 Scatter Plots of Diversity and Fitness

The Spearman correlation coefficient only describes linear relationships. A series of scatter plots are also examined to assess the data for any nonlinear relationships. Figure 4.14 and 4.15 plot a population's performance (best fitness found in the population is plotted along the x-axis, where values to the left are better) versus that population's diversity (on the y-axis). Each point represents a population sampled from a different run, where no run is used twice and 10 populations are

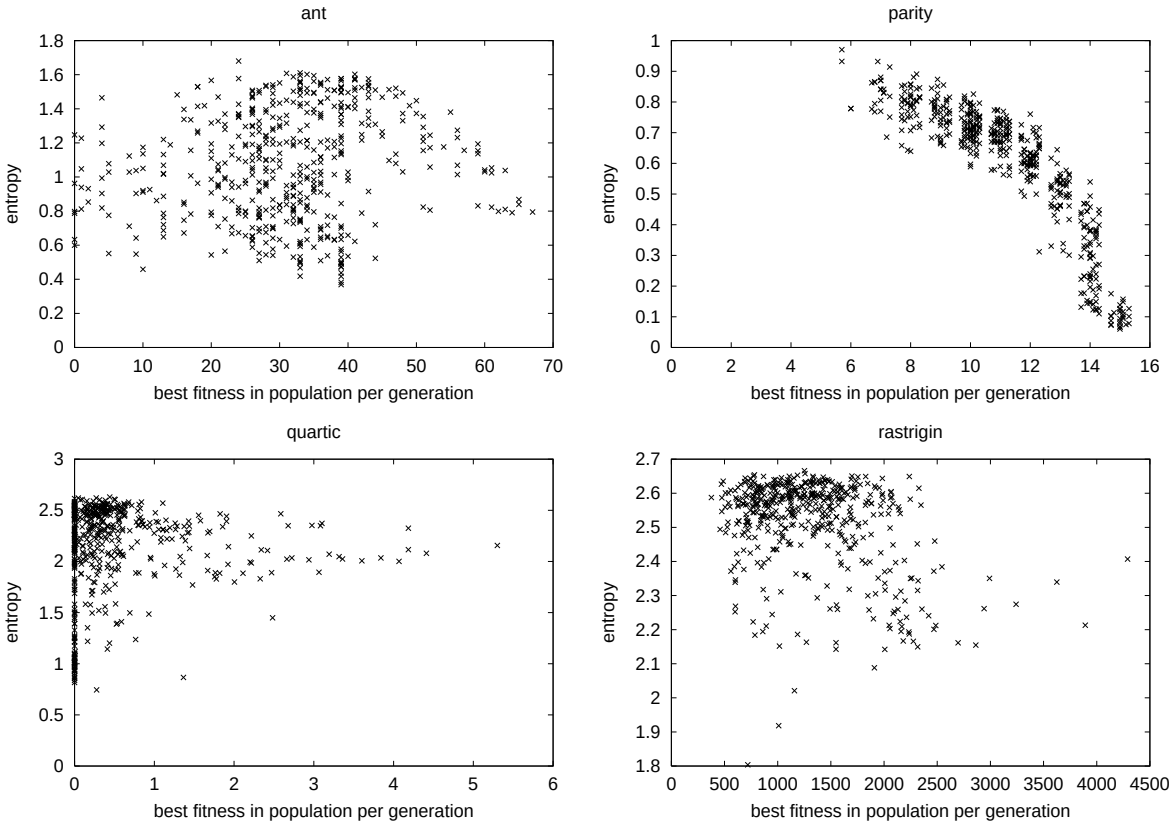


FIGURE 4.14: Ant, Parity, Quartic and Rastrigin best fitness in population plotted against that population's entropy. Note that each point represents one population from each run. We sample 10 different runs for each population at generation g, requiring $50 \times 10 = 500$ runs for all 50 generations.

sampled for each generation, requiring 500 runs. Also note that all points for the Parity experiments have their fitness values randomly offset in the range of [-0.2,0.2] to allow for better visualisation of the points at each fitness value.

A few general comments can be made about the scatter plots in Figures 4.14 and 4.15. There are clear trends of best fitness occurring with lower edit distance and with higher entropy. However, many populations with low fitness also have a wide range of entropy (Rastrigin and Quartic experiments) and edit distance (Quartic experiments). The Ant experiments, in particular, show a transition from high to low fitness with populations in the middle containing a wide range of entropy and edit distance values. The populations which achieve the lower fitness then also have lower entropy and edit distance. It is likely that this problem suffers the most from local optima, where populations that get stuck with sub-optimal individuals also have sub-optimal diversity. Too high edit distance diversity and either too-low or too-high entropy would appear to be sub-optimal for the Ant problem.

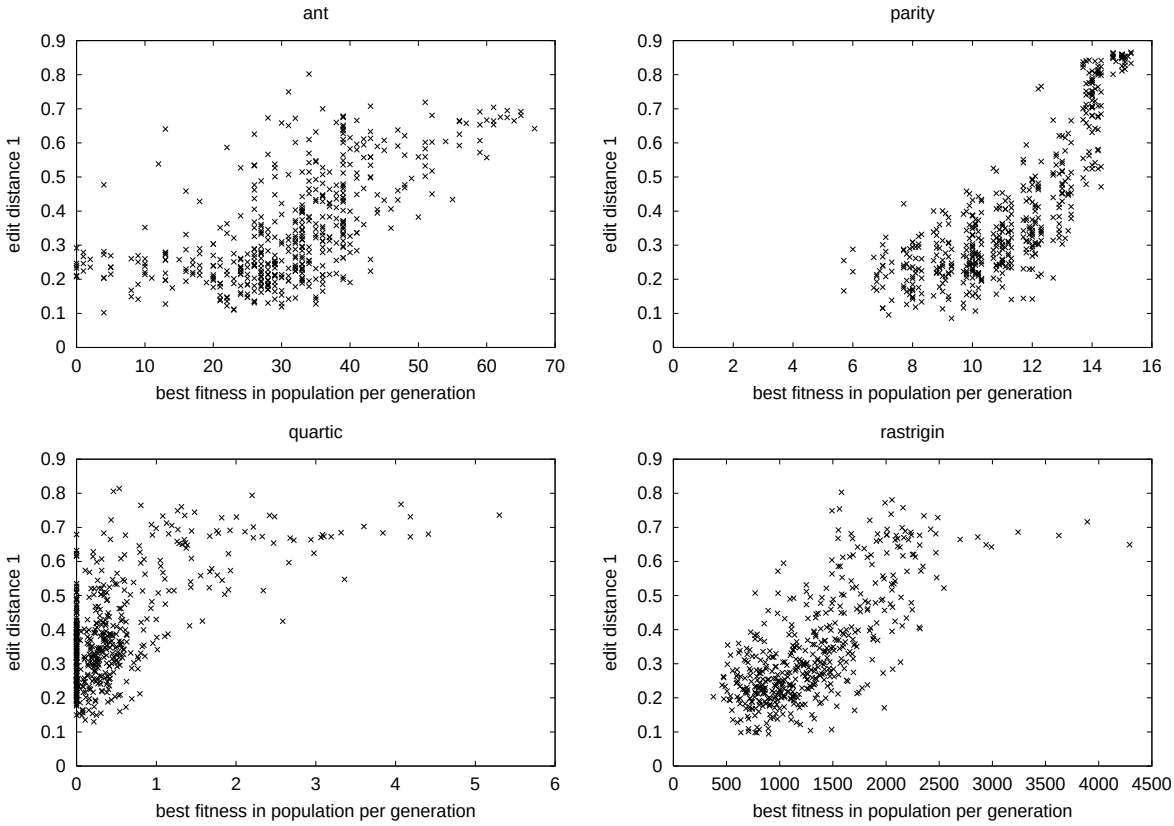


FIGURE 4.15: Best fitness vs. edit distance One diversity for the Ant, Parity, Quartic and Rastrigin experiments. Note that each point represents one population from each run. We sample 10 different runs for each population at generation g, requiring $50 \times 10 = 500$ runs for all 50 generations.

An important observation is that better populations tend to occur near the end of evolution and resulting populations will be less diverse simply because of the search and selection mechanisms. In Figure 4.15, when populations have large edit distances they are unlikely to have better fitness values. A reason for this could be that large edit distances only occur at the beginning of runs. The question of whether these populations always occur late in evolutionary process is analysed next.

For Figure 4.16, the same populations from Figure 4.15 are used, except now the z-axis shows a vertical line representing the generation in which that population occurred. A common trend is that the worse fit populations occur in early generations, which is to be expected as Fig. 4.4 showed fitness to always improve (decrease in value) initially. In general, moving from right to left in fitness values (from worse to better), the lines get taller on the z-axis. However, it is not the case that the best populations are always at the end of runs for all problems. Many populations achieve good fitness early and in the middle of runs. Furthermore, Fig. 4.16 emphasises that populations have different diversity at similar times in the evolutionary process. Later evolutionary periods do not always imply high or low values of diversity and fitness.

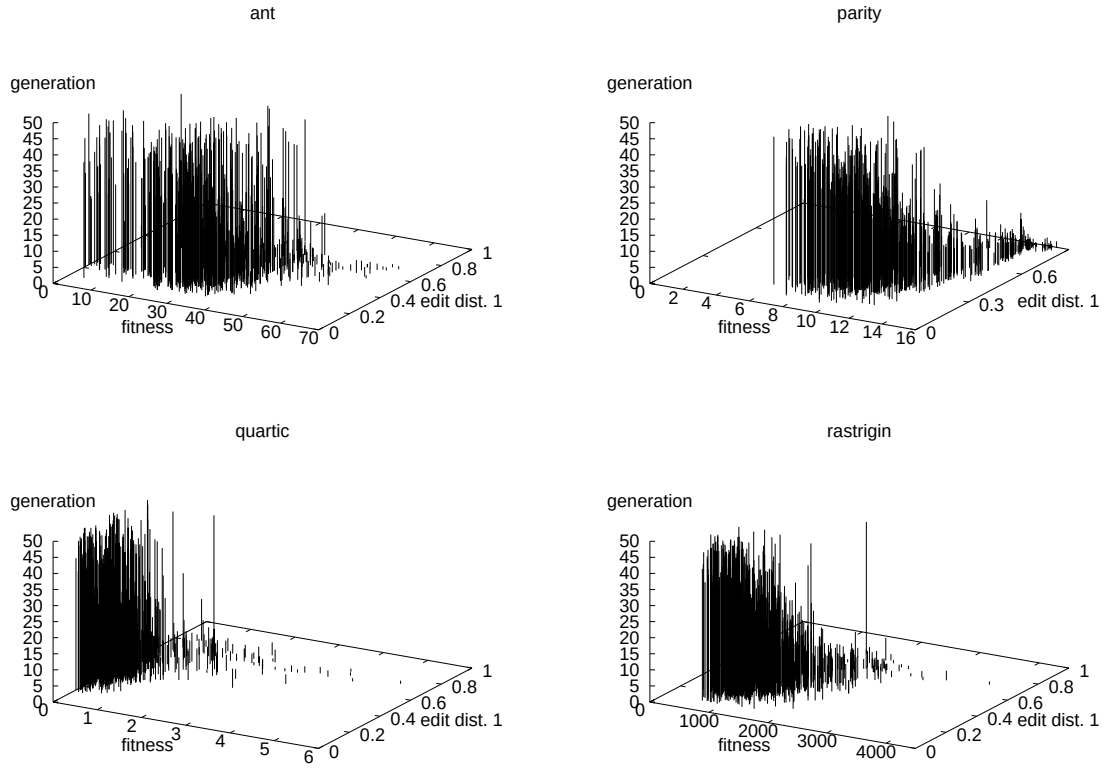


FIGURE 4.16: Ant, Parity, Quartic and Rastrigin best fitness in population (x-axis) plotted against that population's edit distance One diversity, (y-axis) and the generation the population occurred (z-axis). Note that each point represents one population from each run.

4.4 Discussion of Diversity Measures

The measures studied here are in a sense related hierarchically with respect to the amount of information they contain about the population. The edit distance measures provide a fine grain description of population structure and content differences, pseudo-isomorphs give an aggregated view of the population and the genotype diversity measure simply describes the number of unique genotypes. Entropy and phenotype diversity are similarly related. Entropy not only describes the number of unique phenotypes, but also how the population is distributed over the existing phenotypes. Also, the experimental study shows the most consistent correlation between edit distance and fitness and between entropy and fitness, suggesting that these measures capture an important element in the genetic programming search process. The pseudo-isomorph diversity measure was used to capture a level of information that is more specific than genotype diversity, but less expensive than edit distance. Pseudo-isomorph diversity expressed stronger correlations than genotype diversity and is generally more correlated to edit distance measures.

The experiments used different measures of diversity and have enabled the analysis of not only the measures and how they correlate with fitness, but also the behaviour of standard genetic programming on commonly used problems. Results showed additional evidence that the roots of trees become fixed very early on in the genetic programming evolutionary process and are unlikely to change. This has been demonstrated by previous research (Igel and Chellapilla, 1999; McPhee and Hopper, 1999; Soule and Foster, 1998) and is supported here by the edit distance diversity measures.

Phenotypic diversity and entropy are important due to the ability of selection to distinguish between individuals better and maintain a constant pressure. Depending on the problem and behaviour of the current run, the increase and decrease of phenotype and entropy diversity is likely to be crucial at different stages of evolution. This change of selection pressure could be beneficial in helping to avoid local-optima for some problems. The constantly fluctuating values of phenotype diversity in Fig. 3 could be demonstrating this behaviour. However, based on the experiments and analysis, it is not clear if this is necessarily the case.

The Spearman correlation coefficient showed a positive correlation between fitness-based diversity and fitness, and a negative correlation between edit distance diversity and fitness. A hypothesis to explain this behaviour is that more structurally similar populations create a neighbourhood in which crossover is likely to find better neighbours. Crossover initially works with very *unlike* structures until a significantly good one is found. Then, combined with the selection pressure, the population begins to resemble this good individual as crossover repeatedly combines more and more similar individuals. Success at this point suggests that crossover is able to work within this population structure to find better solutions. The results have shown how quickly edit distance diversity is lost. It appears that this crossover-friendly neighborhood occurs early in the evolutionary process, but might also be responsible for leading the search toward inescapable local-optima rather quickly. The point here is not to argue that crossover is (or is not) a sufficient operator for search in tree-based genetic programming, but to show (in cases where genetic programming is solving problems) how populations and recombination operators may be working together.

However, just as the correlation coefficient suggests associations between diversity and performance, it should not be used to infer causation between variants, i.e. higher diversity does not necessarily *cause* better performance but better performance is seen *with* higher diversity (phenotype diversity here). Caution should also be taken considering that the search mechanism's recombination and selection methods play an extremely important role in shaping individuals and populations. Very simple implementation differences can drastically increase or decrease diversity measures. Models of causation based on diversity results should be defined carefully. Later in Chapter 6, lengths are taken to validate a causal relationship, relying on experimental evidence, previous literature and further experimental evidence on a specially constructed model of genetic programming behaviour.

Standard genetic programming is often compared to a random search or a hill-climber, due to the loss of diversity and the attraction to local optima (Gathercole and Ross, 1996; McPhee and Hopper, 1999; Poli and Langdon, 1998a). The results on diversity presented here also support the hypothesis of uneven exploration and exploitation phases. After an initial period of adjustment to different problem representations and selection, the populations appeared to converge toward less edit distance diversity. These initial few generations of each run appear to represent the exploration phase, while the latter part of the run exploited the better individuals. Adaptive controls of diversity, selection pressure or mutations could be used to extend the exploration phase to allow more global search. However, they should also consider the initial *settling* behaviour observed here, which is likely due to the ease of which initial good solutions can be found.

Based on the results presented here, it is hypothesised that strong convergence and exploitation of a common structure occurs in almost all runs, but not all runs exploit a good structure. Thus, genetic programming may be converging to structures which are not amenable to further improvements with respect to the existing population. If the algorithm backtracked upon finding a bad structure, or made a concentrated effort to find a good structure, it could be argued that it is more likely to exploit better structures. In essence, by either increasing the length of exploration or adaptively exploring in later phases, local-optima may be avoided more effectively. Increased population sizes, higher levels of mutation, weaker selection pressure and models which prevent the overall convergence of populations (such as islands, demes or distributed models) could achieve this effect.

4.5 Summary

This chapter has provided a survey of measures used to capture diversity and methods employed to control diversity in genetic programming. An experimental study enabled the analysis of correlation between selected measures of diversity and fitness. The results showed three important behaviours:

1. The generation to generation behaviour of specific diversity measures is problem specific. In fact, representation changes of the same problem are likely to have different diversity behaviours. Thus, the pursuit of a single measure with which to control diversity in order to improve fitness is likely to be difficult.

2. Entropy and edit distance diversity showed some correlation with fitness. This is likely related to a change of selection pressure and the level of structural convergence which allows a form of hill-climbing search.

3. Regression problems had the weakest correlation between any measure of diversity and fitness overall, suggesting that the things that make these populations achieve good fitness may

not be captured by any of the measures used here.

The introduction of different recombination operators, large changes in parameter values and applications on different problem domains are all likely to effect the results and interpretations made here. However, the methodology of using several informative and complimentary measures of diversity should allow one to gain a deeper understanding of the search space and algorithm. As search spaces become larger and more complex, fine grain measures will become too inefficient. Therefore, using measures which capture the right level of information while still being efficient will be critical. Based on these results, the following recommendations are made:

- Before applying new methods to control diversity in order to improve fitness, the correlation between fitness and diversity should be investigated. Knowledge of the correlation between fitness and a measure of diversity can help to enhance the diversity measure or diversity control method and give insight into results.
- Care should be taken to distinguish between correlation and causation.
- Lastly, when a many-to-one relationship exists between the genotype and phenotype encoding, measures which are based on genotype uniqueness will probably not be as useful as those which capture phenotype uniqueness.

Next, a measure of diversity, genetic lineages, is used to increase population diversity (both edit distance and entropy). The effects of this change in population dynamics is examined to better understand the relationship between diversity and fitness, and also the type of search genetic programming carries out during the evolutionary process.

CHAPTER 5

GENETIC LINEAGES AND A METAPHOR OF HILL CLIMBING

What is the effect of increased diversity on search? Chapter 4 showed that diversity can be measured in several often conflicting ways. The relationship between levels of diversity and fitness improvement is clearly not as simple as one would hope.

In this chapter, the genetic diversity in the population is perturbed to increase population edit distance diversity (both measures from Chapter 4) and entropy. A simple but powerful method is employed: lineage selection. After examining the results of this method, a metaphor of hill-climbing is used to support the results presented here and previous results in the literature. Lastly, a sampling analysis is performed to understand the type of solutions genetic programming visits during the course of the evolutionary process.

5.1 Genetic Lineages

In a system based on Darwinian evolution, where inheritance provides the motivation for transformation operators, a simple form of diversity measurement is the genetic lineage. McPhee and Hopper (1999) observed that the dominant operator, subtree crossover, preserves most of the *root* parent's genetic material (also noted by Poli and Langdon (1998a)), and thus they defined genetic lineages between the root parent to its offspring. The authors tracked lineages when they counted the appearance of the *Eve* individual, the latest common ancestor of the population. The *Eve* individual can then be traced back to a single individual in the initial population from which the entire final population descended. That is, when a latest-common-ancestor exists, one can trace the root-parents of all individuals in the final population back to this individual. By definition then, this *Eve* individual is the root-descendant of exactly one individual from the initial population. Remember

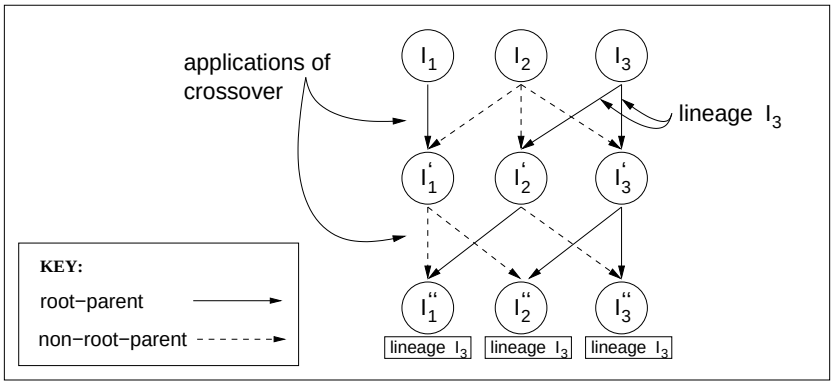


FIGURE 5.1: An example of three individuals undergoing recombination, where after the second application of crossover all three are descendants from the same individual and represent the same genetic lineage.

that the lineage definition only considers the root-parent and offspring relationship.

McPhee and Hopper (1999) showed how entire populations quickly lose many genetic lineages and soon descend from only one individual. As genetic lineages tend to share common root shapes and contents, this loss of lineages in the population signifies the convergence toward a common tree shape and contents (Rosca and Ballard, 1999; Langdon and Poli, 2002; Poli and Langdon, 1998a).

Tracking the progression of genetic lineages through the evolutionary process provides a sense of the loss of genetic diversity in the population at little computational cost. Figure 5.1 is an example of the genetic lineages. Three individuals (I_1, I_2, I_3) produce offspring (I'_1, I'_2, I'_3) via crossover. The root parent is denoted with a solid arrow. After another generation, the three new individuals all belong to the same genetic lineage, I_3. In this chapter, as in McPhee and Hopper (1999), a genetic lineage is defined as the path from the root parent to its child during two parent recombination.

5.1.1 A Caricature of Tournament Selection

To illustrate the loss of genetic lineages, a modified version of tournament selection is used. This version represents a caricature of standard tournament selection, placing emphasis on the selection of the fitter individuals. The individuals in a population p of size μ are sorted into ascending fitness order, from worst to best. A tournament size T defines the number of equal-sized partitions that divide the sorted population, where a partition contains $\frac{\mu}{T}$ individuals. Partitions $p^1, p^2, \ldots, p^T$ serve to separate the worst fit $(p^1 \ldots)$ from the best fit individuals $(\ldots p^T)$. This assumes a unique fitness value for each individual. In a generational algorithm, to produce a child in the next generation, a tournament is held to find the root parent. If the population size is μ, then we hold μ tournaments to produce all the children. Typically, $2 \times \mu$ tournaments are held to select for both

parents for crossover, but since only the root parent and child are used to define lineages, only the one tournament to select the root parent is considered. Every individual in p has the expectation of being selected for a tournament $\frac{T}{\mu} \times \mu = T$ times.

To further simplify the model, tournament selection will pick one individual from each partition, where there are T partitions. Thus, in every tournament held, the winner of that tournament will come from p^T. Every child in the next generation will be of a genetic lineage that came from p^T. In practice, individuals are randomly selected from the population to make a tournament. To simulate the random nature of tournament selection, it will be assumed that the number of offspring each individual in p^T produces for the next generation will be equal.

5.1.2 Variation of Loss of Lineages

It is not possible to increase the number of genetic lineages during the evolutionary process without explicitly introducing new lineages. Genetic lineages can, however, be lost at different rates. In the above model of selection, if the fitness of the root parent is passed to its child. Since every initial individual represents a unique lineage, the next generation will contain at most $\frac{\mu}{T}$ distinct lineages. If the individuals all have unique fitness values that are passed unchanged to children, the ordering by fitness of children will be the same as the parents ordering in the previous generation. Thus, at generation g, the maximum number of distinct lineages will be $\frac{\mu}{T^g}$, with a minimum of 1.

If the fitness of children is not transfered from the root-parent, it is possible to think about upper and lower bounds on the loss of lineages. To consider the upper and lower bounds of genetic lineage loss, remember that the $\frac{\mu}{T}$ members of partition p^T are sorted according to their unique fitness values. The lower bound (or the least) number of lineages preserved in the next generation will occur when the children of individuals from p^T have a fitness ordering consistent with their root-parent ordering. Interestingly, this also holds true when the fitness ordering is the complete opposite of the root-parent ordering. In the former case, offspring produced by an application of the operator to their root-parent undergo small fitness value change from the parent. The upper bound (or the most) number of genetic lineages preserved in the next generation from p^T will be when the children are ordered randomly compared to the root-parent ordering in the previous generation.

Admittedly, this caricature of tournament selection is elitist toward the better individuals, which are found in partition p^T. This behaviour is emphasised to illustrate the loss of lineages, and that in a set of runs, those runs with more genetic lineages are probably performing a more parallel type of search.

5.1.3 Relevance to the Fitness Landscape

If the fitness landscape is smooth with our representation, operator and fitness function, then an application of an operator should cause a relatively small change in fitness. If the landscape is rough, then small changes to an individual by the operator may cause large or small changes in fitness. With a smooth landscape, genetic lineages should be lost quicker. More random changes in fitness between root-parents and offspring occurring with a rough landscape will preserve the most lineages. As it is generally more preferable to have a correlated or smooth landscape, the quicker loss of lineages would be a sign of good search.

However, the loss of lineages also implies genetic convergence and the inability to escape local optima. In canonical genetic programming, using subtree crossover, genetic lineages will begin to share more and more genetic material, from the root downward, during successive generations. As individuals become larger, the size of subtrees inserted into the root parent becomes relatively smaller and at lower points, respectively (Luke, 2003). Quicker genetic lineage loss leads to quicker genetic diversity loss. The evolutionary process then becomes a sort of local search over the converged population's tree shape(s) and contents.

If the operator, representation and fitness function create a rough landscape, genetic lineages will be lost more slowly, more genetic diversity will be preserved longer and more varied tree shapes and tree contents will remain in the population longer. However, this later convergence time and increased diversity is the consequence of a more uncorrelated landscape, which is generally undesirable. Thus, a paradox exists with respect to loss of lineages and ability of improvement. If the operator works 'well', genetic lineages and diversity should be lost rather quickly, getting the algorithm stuck in local optima. If the operator and representation do not work well together and induce a rough landscape, genetic diversity is maintained longer, whereby a more global search may be performed. The loss of genetic lineages is a desirable property as it signifies the correlated landscape in our representation and operator, but it also signifies a loss of diversity which will eventually prevent runs from improving.

Results in Chapter 4 showed that lower genetic diversity was more often correlated with better fitness, supporting the idea that better fitness was achieved when lineages were lost quickly due to a more correlated landscape. The following experimental study serves to further highlight these conclusions about diversity and to provide an analysis of the effects of increased diversity.

5.2 Experimental Study using Lineage Selection

To better understand the effects diversity has on search, a simple method is used based on genetic lineages to increase the genetic diversity of populations. No elitism, size, shape or content bias is added. Three problem domains are investigated and compared with previous research to understand why increasing diversity is beneficial in some domains but not others.

The definition of genetic lineages, described in the previous section, is combined with tournament selection to redirect selection pressure from the *fit* to the *fit and diverse*. As lineages provide an approximation of diversity in canonical genetic programming, this technique does not require a measure of diversity, but it significantly changes populations during the evolutionary process. The goal of this study and technique is to demonstrate that increasing diversity can lead to dramatic changes in the search ability of genetic programming. Population convergence and increased selection pressure of similar individuals creates a 'hill-climbing' atmosphere which, when disturbed, can improve or worsen fitness, depending on the problem. Neither method, genetic programming or hill-climbing, is proposed to be better than the other. Instead, the similarities and differences between the methods are used to explain the effects of changing diversity.

One of the main challenges of search in general is preventing the system from getting stuck in local optima. As highlighted in earlier chapters, convergence is often associated with the inability of the run to improve, but it is also related to the exploitation phase during the evolutionary process. Many problem and representation specific methods have been used to improve diversity. While some methods of diversity show improvement of fitness, they typically add elitism, suffer from additional computation and address a problem which is not clearly defined or understood. How does one know what type of diversity is needed and how much of it is necessary for different problems? As stated by Ryan (1994), "...what is needed is a method which does not attempt to explicitly measure genetic differences, for this leads to much difficulty when defining exactly what constitutes difference". Also, it can be difficult to understand why a problem would benefit from different types and levels of diversity.

5.2.1 Lineage Selection

Lineage selection is implemented as an additional step to bias selection toward different lineages from the initial population. To perform selection, individuals are placed into groups based on common genetic lineages. Tournament selection picks individuals by first picking a random genetic lineage and then a random individual from within that lineage. A tournament is held between these random individuals. Each genetic lineage has an equal change of contributing an individual to each tournament. The method introduces no elitism or no direct measure of size, shape, content,

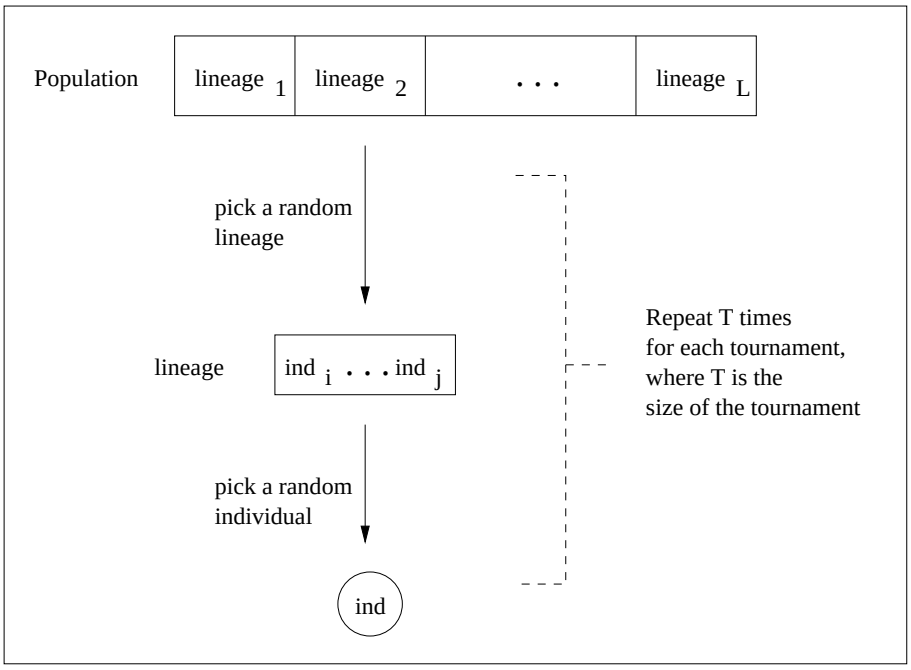


FIGURE 5.2: An example of lineage selection. An individual (ind) is randomly chosen from a randomly selected lineage to compete against T others in a tournament.

or fitness. The aim of lineage selection is to maintain diversity so that the population contains *good* individuals that are not *just* diverse.

Figure 5.2 shows an example of lineage selection, which can be described as follows:

1. Select a random lineage k from $1 \ldots L$, where L is the total number of unique lineages in the current population,
2. Select from lineage k, which consists of individuals $i \ldots j$, a random individual, where $j - i \geq 1$,
3. Repeat steps 1 and 2 above to produce T individuals for a tournament of size T.

Thus, a tournament consists of random individuals from random lineages.

5.2.2 Other Forms of Lineage Selection

Other implementations of lineage selection could allow further control of genetic diversity via genetic lineages. Here, two other selection schemes are described that could be used with genetic lineages to accomplish different objectives.

TABLE 5.1: Experiment and problem parameters for the lineage selection experiments.

Functions	
Ant	if_food_ahead, progn2
Parity	and, or, nand, nor
Binomial-3	+, -, *, p/
Terminals	
Ant	left, right, move
Parity	D1, D2, D3, D4, D5
Binomial-3	x, ERCs
ERC range	$[-10, 10]$
Maximum generations	101

- Select T random individuals from within a random lineage to make a tournament. Now selection gives each lineage an equal chance of participating in recombination. This form of selection promotes the combining of individuals that are from different lineages. A potential drawback is the selection of poor quality lineages where no individuals have high fitness. Thus, it would seem necessary to add a bias toward selecting more highly-fit lineages, or those with a high average fitness.

- Select $2 \times T$ random individuals from within a random lineage to make two tournaments, one for each parent in recombination. This form of selection promotes the recombination of genetically similar individuals and could be likened to a form of hill-climbing within a lineage. Assigning a bias toward more fit lineages, ensuring that lineages have a sufficient number of individuals to perform two tournaments and closely observing the convergence within lineages are areas that may need to be addressed within this form of selection.

These three methods, the one used here and the two above, modify the standard tournament selection scheme to force competition and breeding across lineages, force competition within lineages but breeding across lineages, and finally to force competition and breeding from within the same lineage. The more restrictive the method becomes, the more the algorithm will need to consider the potential drawbacks. However, particularly with the last method, these forms of selection could be considered very efficient implementations of more familiar models of similar and dissimilar mate selection and distributed models.

5.3 Results of Lineage Selection

Lineage selection is applied to the Ant, Parity and Regression problem domains with two experiments: a control experiment with tournament selection and an experiment which employs lineage selection. Table 5.1 describes the parameters used for the following experiments. The Binomial-3 problem is used in Chapter 6 and is described there in greater detail. The other problems and pa-

TABLE 5.2: The Ant, Parity and Binomial-3 statistics for the lineage selection and control experiments. Significant difference is denoted with a '*' next to the mean values for lineage selection. Significance testing was done using the Student's T-test at the 95% confidence level.

The Ant Problem

		Min	Max	Mean	Stdev
fitness	control	0.000	37.000	15.060	12.362
	lineages	0.000	29.000	10.930	10.010
nodes	control	43.408	116.180	79.068	14.878
	lineages	41.968	88.408	62.370*	8.672
entropy	control	0.292	1.169	0.709	0.170
	lineages	0.542	1.509	1.127*	0.235
edit-d	control	0.120	0.353	0.245	0.048
	lineages	0.187	0.365	0.275*	0.036
edit-d (W)	control	0.615	3.572	1.643	0.628
	lineages	1.047	4.394	2.884*	0.711

The Parity Problem

		Min	Max	Mean	Stdev
fitness	control	0.000	13.000	6.740	2.207
	lineages	5.000	11.000	8.970*	1.195
nodes	control	68.064	220.268	124.125	26.762
	lineages	63.136	109.580	82.896*	9.443
entropy	control	0.437	0.969	0.749	0.092
	lineages	0.643	0.940	0.787*	0.05
edit-d	control	0.102	0.409	0.221	0.066
	lineages	0.259	0.471	0.363*	0.042
edit-d (W)	control	0.356	3.494	1.042	0.516
	lineages	2.335	5.490	4.507*	0.580

The Binomial-3 Problem

		Min	Max	Mean	Stdev
fitness	control	0.000	5.480	0.651	0.972
	lineages	0.007	6.930	1.428*	1.875
nodes	control	3.000	141.308	57.351	24.950
	lineages	2.992	84.372	34.401*	21.659
entropy	control	0.287	2.614	1.920	0.554
	lineages	0.264	2.662	1.888	0.819
edit-d	control	0.200	0.533	0.361	0.060
	lineages	0.227	0.711	0.403*	0.104
edit-d (W)	control	0.664	2.078	1.123	0.308
	lineages	0.677	5.134	2.442*	1.042

rameters are used from Chapter 4. The main difference between the parameters used here and in Chapter 4 is the use of only binary functions here. Thus, the syntax trees created in the following experiments, and for the remainder of this thesis, will be restricted to binary trees.

Figures 5.3 and 5.4 show the behaviour of the system for the control and lineage experiments.

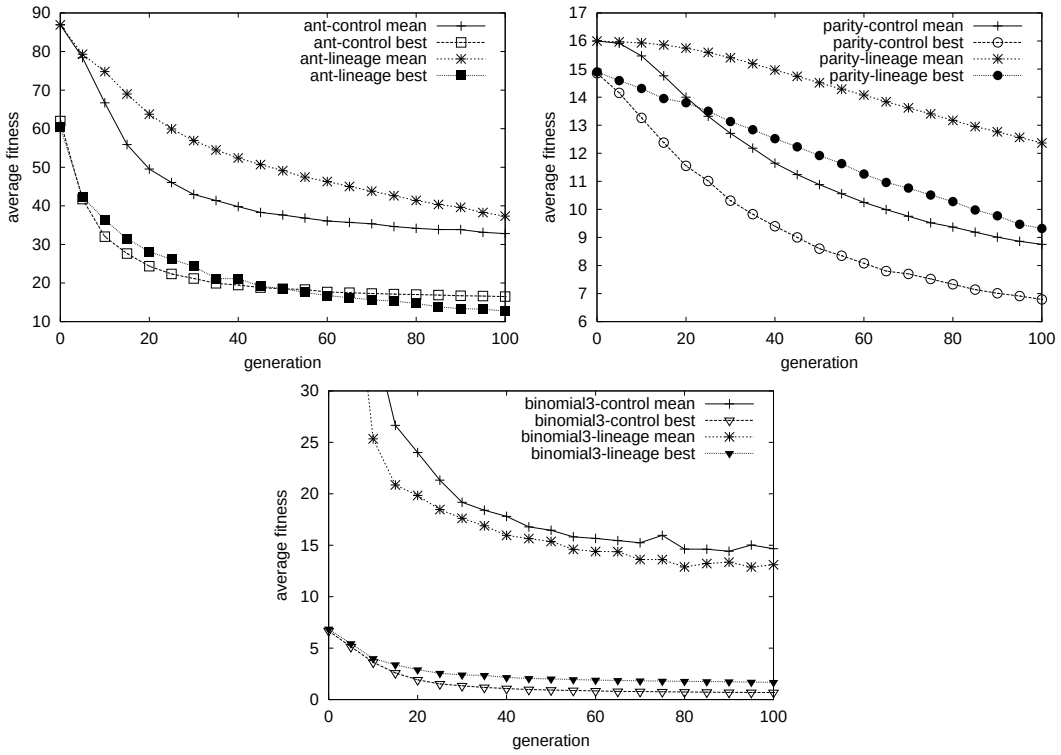


FIGURE 5.3: Average mean and average best fitness vs. generation are shown for each problem and experiment type (control and lineage).

In Figure 5.3, the average best and average mean fitness[1] is plotted against the generation for all experiments. In Figure 5.4, mean run values are plotted for measures of diversity and size, but final generation statistics are also reported in Table 5.2. Note that only the Ant experiments had an improvement in fitness with lineage selection, while all experiments had a significant decrease in size and increase in edit distances using lineage selection. In the control experiment, both measures of edit distance diversity decreased early in the runs and remained low. Initial increases in entropy for the control experiments were followed by either decreases or stagnation. This signifies the inability to improve either the spread of fitness values or the uniformity of the distribution. On the other hand, lineage selection had significantly higher levels of both edit distance diversity. Also, after an initial period of greater decrease of entropy, lineage selection increased entropy longer and to higher values. Figure 5.4 also shows that lineage selection produced significantly smaller individuals.

Figure 5.5 shows that under lineage selection, the distance between successive best fit individuals

[1] It is common, when using 'wrapped' functions (log and division), for solutions in regression experiments to obtain an extremely high fitness value. Thus, when calculating the average mean fitness of the population in each generation, it is necessary to remove these individuals from the calculation to avoid skewed results. This was performed for the Binomial-3 graphs, which resulted in approximately 1% of the population being ignored during the calculation.

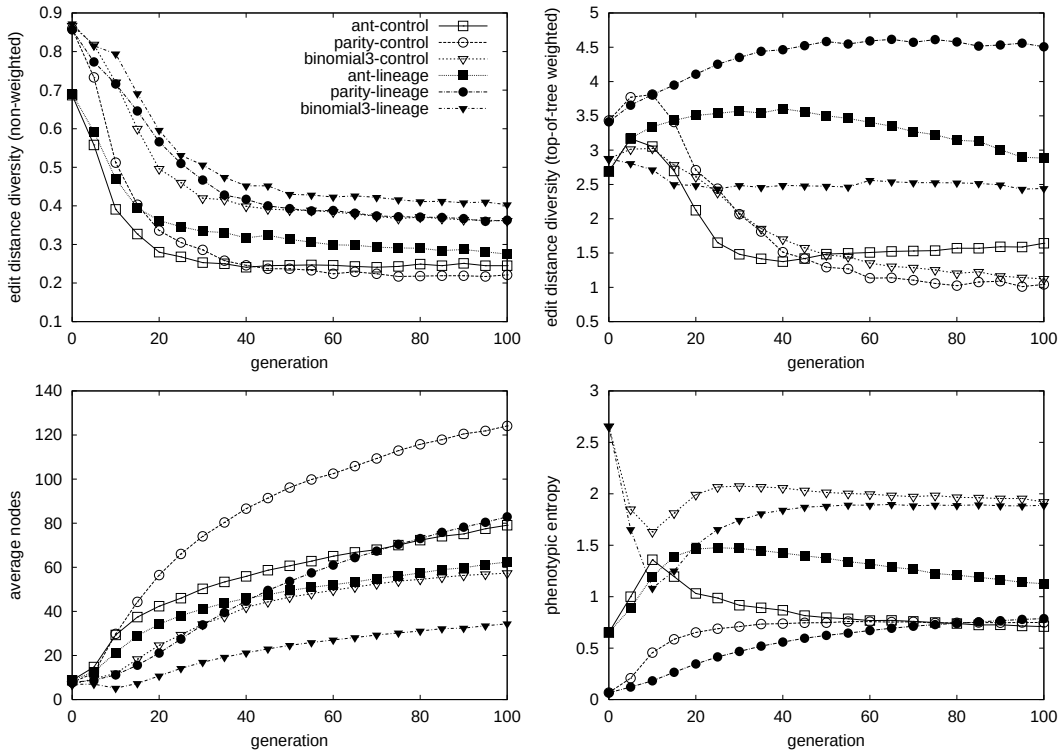


FIGURE 5.4: Average measures vs. generation are shown for each problem and experiment type (control and lineage).

in the population is also higher. Note that the weighted edit distance measure is not normalised by individual size. Because lineage selection produced smaller individuals, this measure was divided by the average individual size to produce a graph similar to the non-weighted measure, but where all the lineage selection experiments remained significantly higher.

The Ant problem was the only one to benefit in terms of fitness improvement from lineage selection. While the fitness for the Parity and Binomial-3 lineage selection experiments were statistically worse, a high level of fitness was achieved in very diverse populations. This behaviour is also reflected in the phenotypic entropy. The difference in entropy values between the control and lineage selection experiments appears to be somewhat correlated to fitness improvement. Only on the Ant problem did entropy stay at the much higher levels after similar initial behaviours. On the other two problems, entropy was much lower in the initial generations (see Figure 5.4). This indicates that the ability to achieve high entropy is hindered by lineage selection in the Parity and Binomial-3 problems, resulting in worse overall fitness. However, in the Ant problem, lineage selection helps to achieve higher levels of entropy and slightly better fitness.

For the Ant experiments, Figure 5.6 shows the last generation where fitness improved versus the best fitness of the run. Under lineage selection, the Ant problem finds better fitness on average

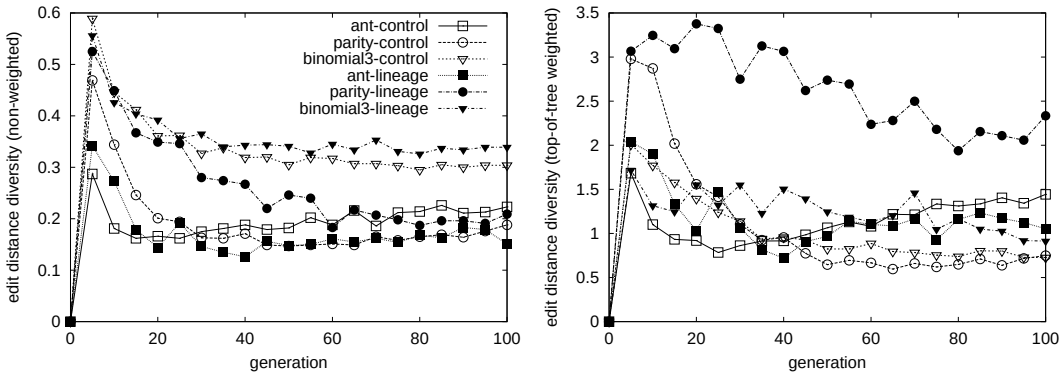


FIGURE 5.5: Edit distance between successive best-fit individuals for the control and lineage selection experiments.

20 generations later than the control experiment. This is a good indicator that premature convergence is being avoided. The Parity lineage selection experiments had a similar change, where the best fitness was found between 10 and 15 generations later but with a slightly worse fitness. The Binomial-3 results were not significant with respect to fitness or the last generation of improvements.

5.4 Discussion of the Metaphor of Hill Climbing

Lineage selection changes the evolutionary dynamics of diversity according to edit distance and phenotypic entropy. The measures of diversity, when increased, are expected to decrease the chance that genetic programming will become stuck in local optima. However, only on the Ant problem did fitness improve. Lineage selection increases diversity by shifting the focus of selection away from the best fit individuals to the fit individuals from different lineages. Note that normal selection pressure is returned after lineages are lost. Why does the Ant problem benefit from reduced selection pressure and added diversity, and why does improving diversity and reducing selection negatively affect fitness on the Binomial-3 and Parity problems? The metaphor of genetic programming performing a type of hill-climbing search is now examined to help understand the results.

In standard genetic programming, the convergence of the population to similar programs leads recombination to be characterised as a type of hill-climber. Thus, one may think of the beginning of a genetic programming run as a short, parallel search period until convergence occurs. At that point, recombination coupled with selection pressure (or elitism) and a converged population behaves like a hill-climber on a single program. If this is considered as a metaphor for standard genetic programming search, then what changes to the algorithm might weaken or strengthen performance?

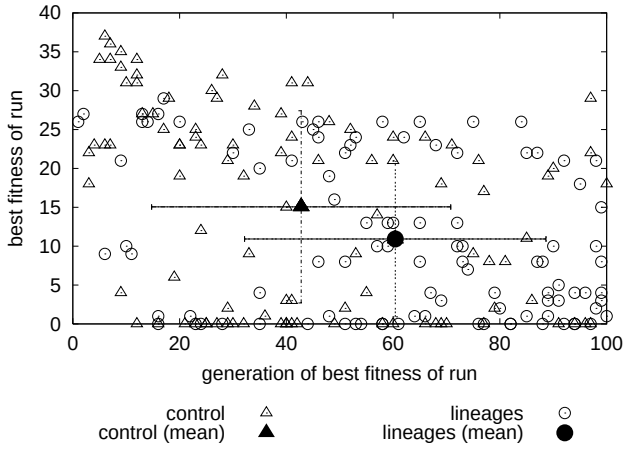


FIGURE 5.6: The generation in the Ant problem where the best fitness of the run was found is plotted against the best fitness of the run. Single standard deviation bars are plotted for both means in all directions.

5.4.1 Artificial Ant

Langdon and Poli (2002) described the Ant problem to be highly deceptive for genetic programming. The problem contains numerous solutions with a lot of symmetry and there is no 'guiding' force to encourage the Ant to travel any particular path. The authors also showed that the Ant problem is solved better using genetic programming than simulated annealing and hill-climbing techniques. Only population based, mutation-only search performed considerably better, as did a variant of strict hill-climbing, which allowed for trees that were smaller and larger than those that would be typically produced by subtree crossover. As population search only uses mutation, it should maintain a high amount of diversity and be similar to performing several hill-climbing searches in parallel. This search method should deal with deception better and not get stuck in local optima as frequently, which explains better performance.

Lineage selection also adds a similar component of parallel search to the Ant problem. High edit distance diversity is maintained and selection pressure is reduced, creating a parallel hill-climbing effect that escapes local-optima better. When an individual becomes stuck in local optima because of deception, a diverse population is likely to contain another individual which is significantly different and allows the run to continue. Lineage selection increases or maintains higher entropy longer with more fitness values or more uniform distributions. In the control experiments, entropy quickly rises and then declines, suggesting a short period of exploration and a higher likelihood of being stuck in local optima. Also, note that in Figure 5.5 the weighted edit distance between best of generation individuals is considerably higher with lineage selection between generations 10-30, the same generations where the entropy values between experiments diverge. The difference

between the best individuals in this phase is found closer to the root with lineage selection. A more explorative search phase appears to be taking place with lineage selection.

5.4.2 Parity

O'Reilly and Oppacher (1994,1995) and Juels (1995) studied the Multiplexer problem with genetic programming and similar hill-climbing type methods . The Parity and Multiplexer problems have similar functions, terminals and objectives, though they are not identical. Hill-climbing techniques appeared superior in this type of problem. Could genetic programming improve performance by becoming more of a hill-climber? De Jong et al. (2001) used a multi-objective method that kept only non-dominated individuals according to an individual's fitness, size and diversity to produce superior performance over genetic programming on the Even-5-Parity problem. Diversity was based on an edit distance between trees. Small populations were used that kept all non-dominated individuals. The authors noted that diversity is required to prevent convergence resulting in run failure with their "uncommon degree of greediness or elitism". Hill-climbing appears to perform well on this problem as does a multi-objective method which simulates hill-climbing.

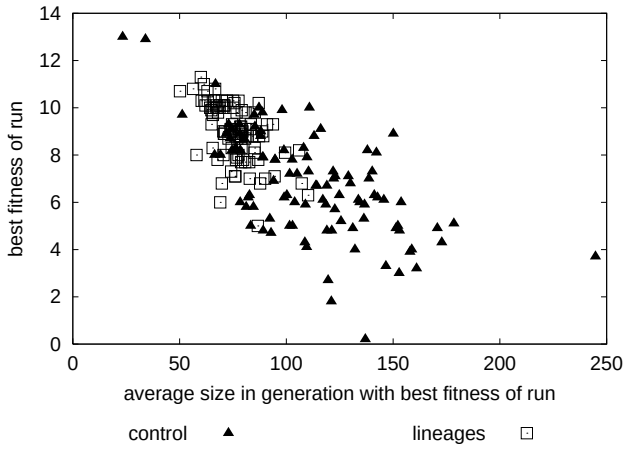


FIGURE 5.7: Average size of an individual in the generation where the best fitness was found in the Parity experiments.

Lineage selection on this problem decreases selection pressure and prevents the loss of diversity and convergence. Lineage selection appears to remove the attributes of genetic programming which allow it to behave like a "hill-climber". Additionally, the size of individuals is significantly reduced under lineage selection, as seen in Figure 5.4 and Figure 5.7. The latter graph shows the best fitness of each run plotted against the average size of an individual in that generation. Only in the Parity problem was there such a distinctive increase in size associated with an improvement in fitness. It is hypothesised that an additional factor is responsible for poorer fitness under lineage

selection. By using code-growth reducing methods, Luke and Panait (2002a) showed that reducing the size of solutions in Parity and Multiplexer problems (in both size restricted and unrestricted spaces) had the effect of also worsening fitness when compared with standard runs . In both problems, the solving of all the fitness cases requires the use of all the terminal values. For example, in the Parity domain, the absence of one of the boolean variables from a program would result in only half of the test cases being potentially solved. There is a benefit to programs which contain several copies of each terminal to increase the chance that they are used properly.

Adding additional elitism or more computation time with lineage selection should improve the fitness results. Also, for problems where it is known that solutions will need particular terminals or functions, it would make sense to encourage their inclusion.

5.4.3 Binomial-3

Daida et al. (2001) provided a thorough investigation of why increasing the ephemeral constant range makes the Binomial-3 problem 'harder'. The authors draw attention to the inter-play between content and context of functions and terminals in the representation. Many different solutions exist to the Binomial-3 problem and combining parts of different solutions does not always make sense. A level of deception exists that is similar to the Ant problem, due to the many different solutions. However, in the Ant problem the functions and terminals preserve, to some degree, semantic meaning in different contexts. Moving constants and arithmetic functions between programs in Regression problems does not ensure their meaning in new contexts. A DAG (directed acyclic graph) representation of genetic programming was used on a Regression problem (Monsieurs and Flerackers, 2003) where the author introduced a diversity method that was also highly elitist. Performance showed that best fitness was achieved much faster with smaller population sizes using the elitist diversity measure.

Regression problems appear to pose a two-fold problem, finding a good approximation to fit the data points and attempting to reduce semantic changes of nodes during crossover. In this case, increasing genetic diversity could increase the chance that crossover will have problems with nodes changing context. A converged population may contain fewer nodes but with similar contexts and improve search performance. However, too little or too much selection or diversity would cause problems as well, making this a complex problem domain.

5.4.4 Remarks

Is increasing diversity beneficial to genetic programming? The results have shown that increasing the genetic differences in populations allows for more global search and local optima avoidance.

The results have also shown that higher genetic diversity leads to less code growth, and that increasing genetic diversity adds longer periods of entropy increase. While higher entropy indicates a more explorative search, lower entropy results in less selection pressure as more individuals have the same fitness value. Also, the slower increase of entropy and higher genetic diversity appears to decrease the hill-climbing behaviour that previous research has shown to be effective in solving these problems. Future methods used to increase diversity to improve fitness should clearly state the motivation for such an increase and why that type of diversity would be beneficial. Diversity methods may not be justified in their own right, but they work together with more elitist strategies or as a supplier of programs to a local search method.

Genetic programming searches for solutions to a given objective using program-like representations. However, the task of evolving both the structure and content of a solution is complex and difficult to understand (Daida et al., 2001; O'Reilly, 1998; Hu et al., 2002). The research presented in Chapter 4 and earlier in this chapter has focused on understanding the relationship between diversity and search, particularly on the kind and level of diversity that encourages good performance. So far this chapter has showed how increased diversity negatively and positively effects performance on several problems. A metaphor of hill-climbing search helped explain the results, where deception appeared to be a cause of poor performance. To further understand diversity and search, particularly with respect to the solutions the population contains during search, the type of structures (tree shapes) and behaviours that genetic programming samples during the evolutionary process will now be examined.

5.5 Sampling of Unique Structures and Behaviours

Research into code growth and operator biases can be used to help understand the type of tree shapes sampled by genetic programming. Subtree crossover and the representation predispose solutions toward code growth and bloat (Soule and Foster, 1997; Langdon et al., 1999; Langdon, 1998b; Banzhaf and Langdon, 2002; Langdon, 2000a; Soule and Heckendorn, 2002; Luke, 2003). While programs continue to grow, they tend to grow toward deeper and less-bushier trees. Also, the space of tree shapes visited during genetic programming search has been studied (Soule and Foster, 1997; Langdon, 2000a; Langdon, 1999; Daida, 2002), showing that there are types of shapes that are more easily sampled than others. If a problem's solution is not within the more easily sampled structures, the problem will be difficult for genetic programming (Daida et al., 2003b).

With respect to the growth of solutions, the subtree crossover operator is shown to be a more "local" operator, where the upper-portion of trees become fixed and variations mainly occur near the leaves (Rosca and Ballard, 1995; Igel and Chellapilla, 1999; Poli and Langdon, 1998a; D'haeseleer and Bluming, 1994). Diversity research also demonstrates the effects of the convergence of structures

(McPhee and Hopper, 1999), and in Chapter 4. However, it is also the content of trees (functions and terminals) that provide a solution to a given problem. By using stochastic sampling techniques, the proportion of solutions of increasing size was shown not to increase beyond a certain threshold (Langdon, 1999; Langdon and Poli, 2002). That is, the space of increasingly larger solutions did not yield a higher proportion of solutions.

As the fitness function is typically a very coarse description of behaviour, it is more difficult to understand the type of solutions sampled by genetic programming. However, comparisons between genetic programming and hill-climbing methods using similar representations and operators can be helpful (Langdon and Poli, 2002; O'Reilly and Oppacher, 1994; O'Reilly and Oppacher, 1996; Juels and Wattenberg, 1995). While genetic programming performed better and worse on various problem domains, comparisons emphasised the domains in which more hill-climbing or explorative search is beneficial. These results were particularly useful in explaining the effects of increased genetic diversity, as seen earlier in this chapter. In any case, much of the solutions' *behaviour* remains hidden behind the fitness function value.

To better understand the sampling of tree shapes and behaviours during search, this section examines the number of unique tree shapes and behaviours (an enriched definition of fitness) sampled. The structure aspect of the following study provides additional views of the search process, while the coarseness of fitness function values is addressed with problem-specific behaviour descriptions that reflect fitness but elucidate the behaviour of the solutions better.

5.5.1 Problems and Measures

The same three problems from earlier in this chapter (from the **control** experiments) are used in the following empirical study. However, in the following study, problem specific measures of solution behaviour are used.

In an evolutionary algorithm, a scalar value is typically used to define a solution's behaviour (although multiobjective methods may use a vector). This value, defined by a fitness function, must be able to distinguish different degrees of solution quality. This coarseness often leads to deception, as in the Artificial Ant problem (Langdon and Poli, 2002), or fails to identify solutions that are relatively good in the current population but extremely poor to continue a search with, as in the case of very small solutions in Regression problems. Thus, measuring the sampling of behaviours during a run using only the fitness value may not be as informative as one would like, and so problem specific definitions of behaviour are used, defined as follows:

- *Ant*: each food item is uniquely labeled to create a vector representing the order in which the food is collected,

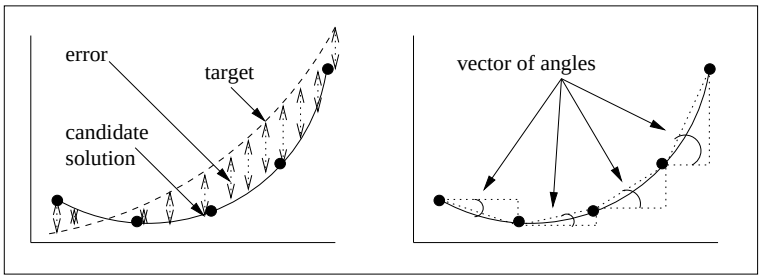


FIGURE 5.8: Behaviour definition example for the Regression domain, with standard fitness calculation (mean squared error) and the behaviour definition.

- *Parity*: a vector of integer values, where the size is the total classified correctly and the contents indicate which of the 2^5 test cases were correctly classified,
- *Regression*: a vector of integer values represents the angles between test points (from the horizontal) taken by a solution.

These definitions capture more information than their typical scalar value fitness would, but are still a reduction of the complete behaviour of a solution. The Ant behaviour represents the unique sequence of food collection, where the largest sequence is 89 and the smallest is 0. The Parity behaviour describes which specific instances are correctly classified. The definition of behaviour for the Regression domain describes the change in angle from the horizontal between each function point in the candidate solution. By casting these angles as integers, when two successive angles in the vector are identical, only the first is kept. Thus, the size of the vector alludes to the complexity of the solution (the number of "bends" in the graph of that function), but not directly to the fitness value as the target function is not considered. Figure 5.8 highlights the differences between the standard fitness function and the definition of behaviour proposed here.

In this section, the tracking of sampled structures is performed by considering the binary tree shapes regardless of tree content. The number of unique tree shapes of each size that are sampled during the run of the algorithm are counted.

A canonical genetic programming system is run for 51 generations, using a generational algorithm with a population size of 500. Initial tree creation is carried out using ramped half-n-half with tree sizes between depths 2 and 4. Subtree crossover, with internal node selection set at 90% probability and maximum depth of 10, is used for recombination – no mutation or duplication is used. There are 30 random runs collected for each problem domain.

The results for each problem domain are depicted in two graphs showing the average number of unique tree shapes sampled and unique behaviours sampled of a given size. Each line represents the cumulative total of unique tree shapes (or behaviours) in each generation during a run. Each

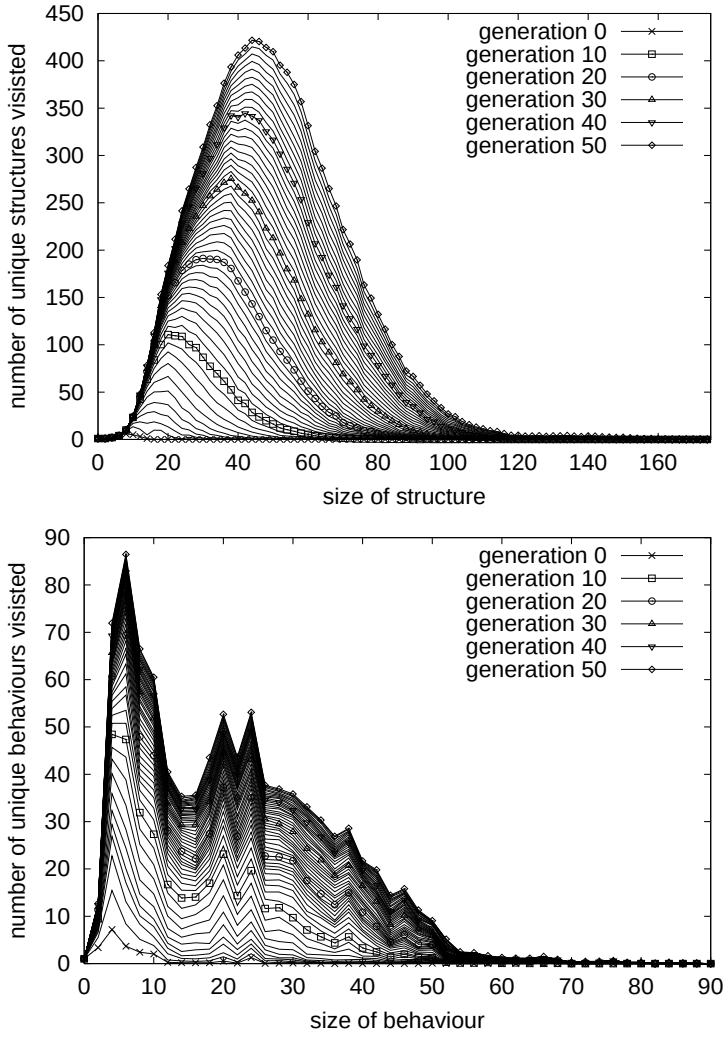


FIGURE 5.9: Ant results, cumulative sampling of unique structures and behaviours.

successive ten generations are highlighted with symbols. Thus, the space between two lines represents the number of new unique tree shapes (or behaviours) of a given size that were sampled in a generation. The larger the space, the more effort genetic programming spends on searching unique tree shapes (or behaviours) of that size.

5.6 Analysis of Results

Figure 5.9 shows the structure and behavioural sampling for the Ant problem. There is a distinct trend of the highest number of unique structures sampled toward sizes of 45. The bottom graph in Figure 5.9 shows the sampled behaviours of each size for the Ant problem. Note that the number of

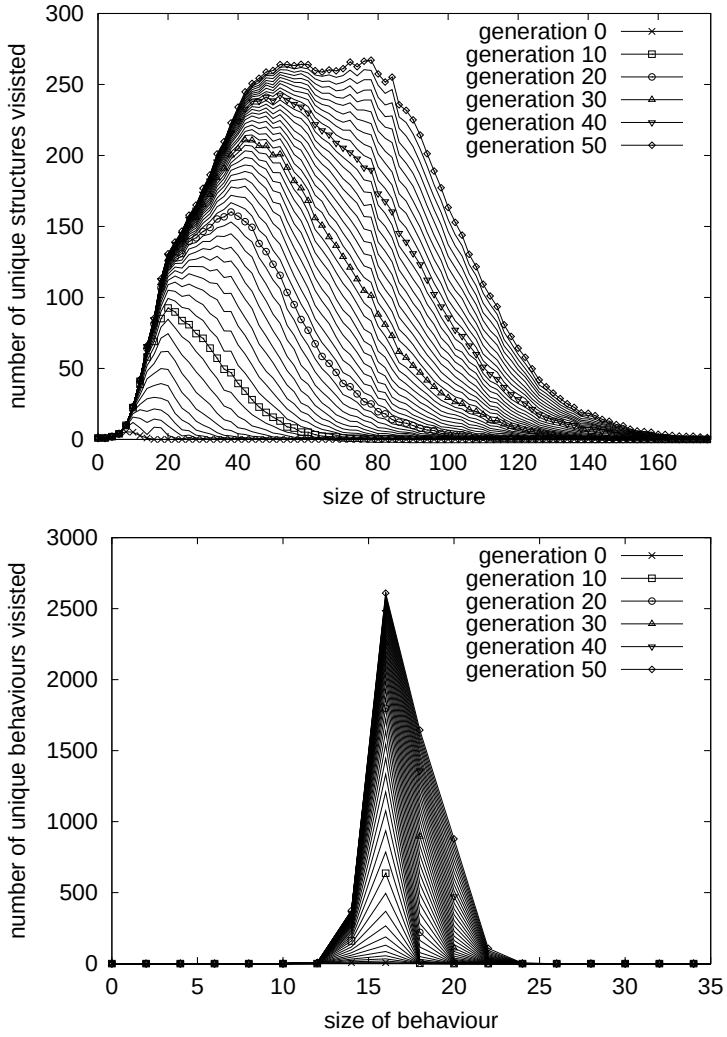


FIGURE 5.10: Parity results, cumulative sampling of unique structures and behaviours.

unique behaviours sampled of all sizes in each generation is greatly reduced after approximately 10 generations. Also, after two unusual peaks at behaviours of size 20 and 24, the number of unique sampled behaviours greatly decreases for behaviours of large size (which represent more "fit" solutions).

The sampling of structures for the Parity problem is shown in the top graph of Figure 5.10. This problem samples fewer unique structures but at larger sizes. The number of unique behaviours sampled in the Parity problem are shown in the bottom graph of Figure 5.10. A behaviour has a maximum length of 32, which represents all 2^5 correct classifications. However, there are $\binom{32}{k}$ possible unique behaviours for a length of k. For the expected random strategy classification of size 16, nearly 2500 unique behaviours are sampled over the course of a run. Genetic programming spends

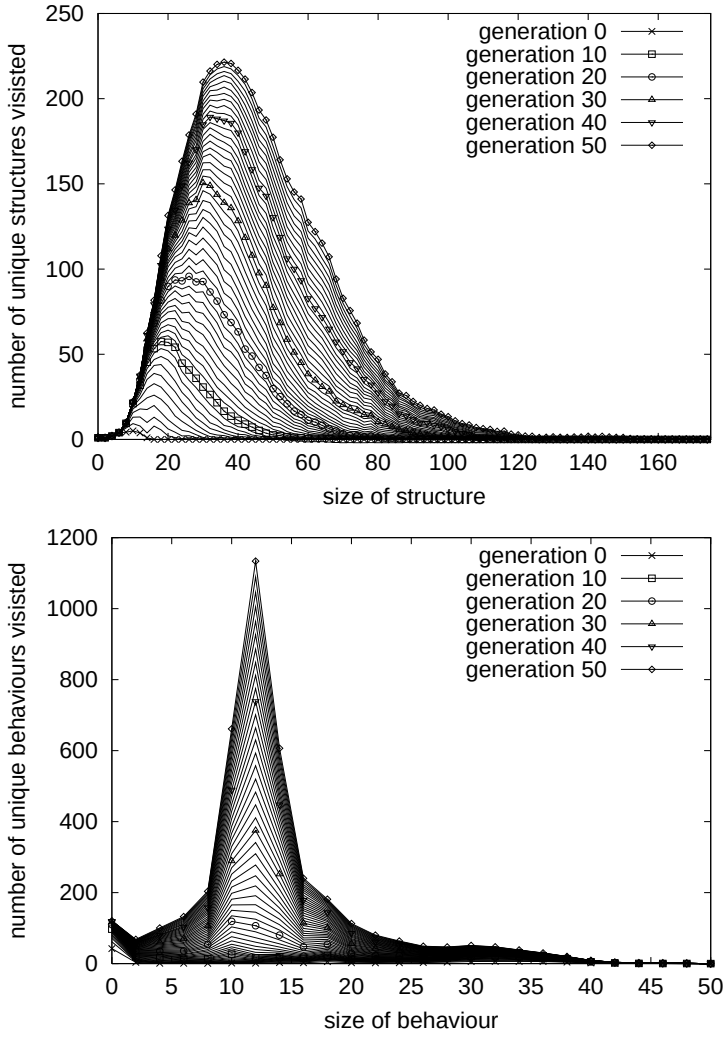


FIGURE 5.11: Regression results, cumulative sampling of unique structures and behaviours.

a large amount of effort searching neutral behaviours equivalent to a random strategy, with slight peaks at fitness 18 and 20. Symmetry in the Parity bit-string instances probably rewards the solving of an additional instance with another symmetrical instance also solved correctly, explaining why fitness is concentrated on the random (16) strategy initially, followed by one instance additionally solved (17+1=18) and then another (19+1=20).

The Regression problem's sampling of structures is shown in the top graph of Figure 5.11. A distinct trend is seen toward sampling unique structures of sizes near 40. Fewer unique structures of larger sizes are sampled. The number of unique behaviours sampled in the Regression problem, depicted in the bottom graph of Figure 5.11, shows a strong attraction toward behaviours of size 12. All generations during the run sample unique behaviours of this size. As behaviour does not directly

reflect the fitness, these behaviours may or may not have neutral fitness. However, the Binomial-3 fitness function contains 50 equidistant x values that generate the target y values for testing an individual. The angle gradient between successive y values is nearly always greater than 1. Thus, for our behaviour measure, if a behaviour is to represent the function ideally, it will need close to 50 angle changes between points.

The 95% confidence bars for each of the average distributions from Figures 5.9 to 5.11 are shown in Figure 5.12. From left to right, the Ant, Parity and Regression structures are shown on the top row, and behaviours on the bottom row. The sampling between the 30 runs is fairly uniform with the greatest variations occurring near the peaks in the Regression problem.

5.7 Discussion of Sampling

How does one search for computer programs, and how does one know if one program is better than another? The intuitive broad sampling of a complex solution space by a population in genetic programming is one reason why the algorithm may be considered useful. Early work by Cramer (1985) and Koza (1992) laid the foundations for this procedural representation, where the use of single-value scalars as fitness combined and complex representations were already in use in other evolutionary algorithm domains. While it may have been straightforward to apply these methods to program evolution for genetic programming, the conflicts arising in this representation are well documented (Daida et al., 2001; O'Reilly, 1998; Daida et al., 2003b; O'Reilly and Goldberg, 1998). The results presented here, based on the sampling of unique tree shapes and unique behaviours, further describe genetic programming's ability to sample a complex solution space.

The behaviour definition has enriched the description of the sampling of solutions by genetic programming. In the Ant problem, the geographic distribution of food pellets on the Ant trail may allow for many Ant behaviours that are able to collect 20 to 24 pellets, but unable to easily collect more. In this problem, the standard definition of fitness creates deception (Langdon and Poli, 2002) that has already been shown to negatively impair fitness improvement. However, the more gradual decrease in the ability to sample different behaviours of higher fitness explains why genetic programming often outperforms hill-climbing methods that search with poor solutions. The peaks of behaviours sampled at size 20 and 24 may represent local optima that trap search. The behaviour distribution for the Parity problem is particularly interesting as it shows the ability of genetic programming to sample many different near-random type behaviours. Sampling such high numbers of different behaviours is promising but a likely cause of deception and negative contextual shifts of subtrees, explaining why hill-climbing and elitist strategies are more effective on this problem (O'Reilly and Oppacher, 1994; Juels and Wattenberg, 1995).

While the Regression problem used a behaviour that did not directly reflect its fitness value, the behaviour did reflect the complexity of solutions. The preference toward sampling behaviours of a small size and complexity could be caused by neutrality in the fitness values or the result of a biased structure sampling. In the latter case, the inability to increase structure size is hypothesized to be positively correlated with the inability to increase the complexity of solutions. This hypothesis seems plausible for the Regression domain, considering that subtree crossover typically functions near the leaves (terminals).

With respect to the sampling of unique structures of various sizes, all problems showed that while genetic programming is predisposed to code growth (and bloat), unique tree shapes of large sizes are sampled less. That is, a fewer number of big tree shapes are sampled. Remember that the number of different programs for a given tree shape increases exponentially with an increase in size. Thus, while fewer large and unique tree shapes are being sampled, the algorithm could be sampling more different programs of those shapes. The different structures of the same size that are sampled more frequently would appear to be attractive to genetic programming for one reason or another. The depth limit imposed here and the mechanics of subtree crossover are also possible causes.

Poli and McPhee (2001) examined the distributions of a variable-length linear genomes using standard crossover and a flat landscape. The authors found that the distribution of genome lengths sampled during search resembled a gamma distribution and that shorter genomes were sampled much more than longer ones. With respect to the results presented here, when the non-cumulative version of Figures 5.9 to 5.11 are examined, the distribution of unique tree shapes sampled in the Ant and Regression experiments also resembles a gamma distribution. While it is less clear for the Parity experiments, additional analysis into these distributions would provide further support of

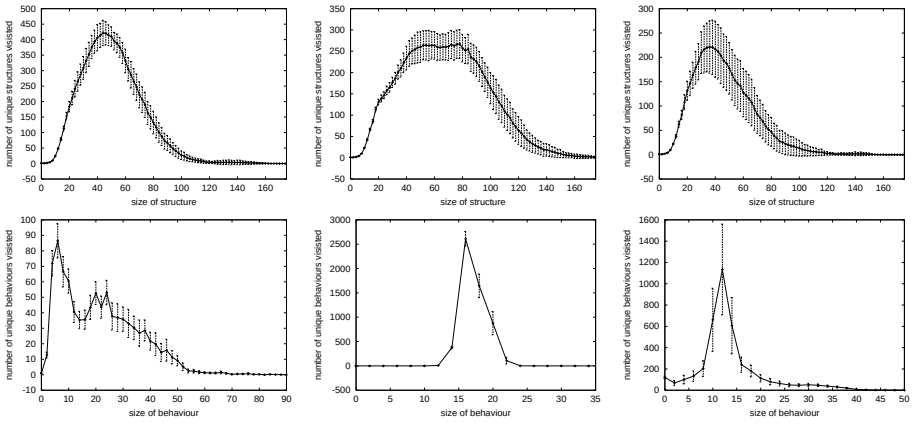


FIGURE 5.12: 95% confidence bars for the average cumulative structure and behaviour sampling distributions. From left to right is the Ant, Parity and Regression problems with structure on top and behaviour below.

the results presented in (Poli and McPhee, 2001).

These results also suggest several possibilities to improve performance. The sizes of behaviours that were sampled in high numbers may represent areas of deception in the fitness space. This deception may be reduced by pressuring the search away from areas of neutral fitness. Previous research showing a positive correlation between high fitness-based diversity with good fitness could be the result of avoided deception leading to better fitness. Thus, when a population contains many different but equal in fitness behaviours, we may wish to move the search toward a less deceptive region of the search space. Leading the ant to follow a specific food trail reduced deception and improved fitness (Langdon and Poli, 2002). For the Regression domain, using a component of complexity in the fitness function may also improve the performance of genetic programming. Similarly, the Parity fitness function could be amended to reflect which instances a solution solves correctly to reduce deception. Of course, exploration (global search) and exploitation (local search) ability will be affected when such methods are employed to reduce deception.

5.8 Summary

This chapter has described how genetic lineages can serve as a simple measure of diversity in a representation based on natural selection. Many possible uses exist in genetic programming for lineages, here a variant of selection was used to further describe the search process. The relationship between genetic lineage loss, diversity and fitness landscape was described. A correlated landscape, where the application of an operator causes small changes to fitness, will encourage the quicker loss of genetic lineages and diversity. A uncorrelated landscape will lose genetic lineages and diversity at a slower rate.

Lineage selection is used to increase diversity by reducing the selection pressure from the most *fit* to the *fit and diverse*. This has caused performance variance across three problem domains. The results were analysed in the light of previous research to conclude that, if genetic programming is viewed as performing a type of hill-climbing search, adding diversity can worsen fitness on some problems that clearly benefit from elitism in a hill-climbing environment. However, when deception is embedded into the problem, improving diversity may help avoid local optima (as in the Ant problem), or it may compound the deception by maintaining its presence (as in the Binomial-3 problem).

The last section of this chapter examined the sampling of unique tree shapes and unique behaviours in genetic programming. An enriched definition of behaviour provided more information about solutions than typical fitness functions. As the genetic programming algorithm requires the search of structure and content, it is important to understand issues such as deception and the effort the

algorithm spends on searching different types of behaviours and structures. The behaviour sampling results showed sampling trends that help explain previous diversity research and suggests new ways to improve search.

Also, the sampling results showed that there are different behaviours with the same fitness values in all problems that genetic programming samples at much higher rates. If low phenotype diversity and entropy are likely indicators of deceptive regions of the search space, and considering the results from Chapter 4 showing a correlation between high values of these measures and better fitness, then better search is achieved when deceptive regions are avoided. Thus, adaptive measures that recognize the signs of deception could possibly help to improve search. Lastly, the structure sampling results showed that while bloat and code growth occur, fewer different unique tree shapes of these large sizes are sampled. Problems that require specific structures at these large sizes are likely to be more difficult for genetic programming.

In the next chapter, the relationship between diversity and other aspects of the search processed is examined. In particular, the effects of population diversity are examined with respect to code growth and problem difficulty.

CHAPTER 6

EFFECTS OF POPULATION DIVERSITY: CODE GROWTH AND PROBLEM DIFFICULTY

The issues of code growth and problem difficulty were introduced in Chapter 3. In Chapter 4, various forms of diversity were shown to have a complex relationship with fitness improvement, where edit distance and fitness-based measures seem to represent important properties. Chapter 5 furthered the analysis of diversity by illustrating expected diversity loss using the concept of genetic lineages and a caricature of tournament selection. Also, Chapter 5 highlighted a potential metaphor of genetic programming search and the signs and consequences of deception in the search space. How do these results apply to important issues such as code growth and problem difficulty?

In this chapter, diversity is closely examined with respect to problem instances that are both tunably difficult and that exhibit varying rates of code growth. The results strongly support a causal hypothesis relating code growth and diversity. This hypothesis is also supported by previous literature and further experiments using a constructed model of code growth and problem difficulty. While furthering the understanding of diversity and problem difficulty, this chapter also provides important insights that suggest new ways of addressing the issue of bloat.

6.1 Code Growth and Problem Difficulty

Two challenging problems in genetic programming are the application of genetic programming on increasingly difficult problem instances and the increase of solution size which is not correlated with improvement. Daida et al. (2001) recently demonstrated that when genetic programming has difficulty solving harder problem instances, solutions are generally larger in size, take longer to find, and are less numerous in the population. Of these traits, the increase in size without corre-

sponding fitness improvement is the most worrying to genetic programming practitioners. The continued growth of solutions for difficult problems will become a limiting factor of the applicability of the algorithm; computational resources will be exhausted and the algorithm will halt before reaching a good solution. The existing theories of code growth (Langdon and Poli, 2002; Luke, 2003; Soule and Heckendorn, 2002) provide the mechanical basis for understanding why programs in a variable-length representation tend to grow in size, but do not predict the rate of growth or explain why optimal solutions can vary in size by a large degree. Additionally, the many methods used to limit code growth, or to remove non-functional code from solutions, may often lead to worse performance (Ekárt, 2000; Luke and Panait, 2002a; Soule and Foster, 1998; Zhang and Mühlenbein, 1995).

Two symbolic regression problems are used that are differently scaled in difficulty to observe the behaviour of fitness, code growth and diversity. If growth is related to instance difficulty, it will be likely that the dynamics of the population, viewed with measures of diversity, will aid in understanding the phenomenon. Symbolic regression problems are used for two main reasons. Firstly, there is a large body of existing theoretical studies using regression problems to understand genetic programming. Secondly, it is easier to reason about increasing the difficulty of fitting mathematical functions than it might be for increasing the difficulty of other types of problems.

6.2 Regression Problems and Increased Difficulty

Building on the study by Daida et al. (2001), the Binomial-3 problem with varying ERC ranges is used to increase difficulty. As in Chapter 5, the Binomial-3 problem consists of approximating the function $f(x) = (1 + x)^3$ using the functions $\{+, -, p/, \times\}$ and the terminals $\{x, R\}$. R are ERCs in the ranges of [-1,1) (referred to lated as *unity*), [-10,10) (*ten*), and [-100,100) (*'hundred'*), where larger ranges create increasing difficulty. The Binomial-3 function is shown graphically over the interval [-1,0] in Figure 6.1 (top left). There are several solutions to this function which can be represented easily by many different combinations of functions and terminals. The easiest version of this problem would be expected to allow for many close-to-optimal initial solutions and more new optimal solutions to be acquired in every generation. Harder instances are likely to cause more difficulty in finding good solutions early and often.

To complement this problem, a method is used to generate random polynomial instances (Ekárt and Németh, 2002). Random polynomials are generated up to degree 11 with non-zero coefficients and with real roots in the range $[-1, 1]$. As higher degree polynomials have a larger number of roots close to zero, most of the values they take in the $[-1, 1]$ interval are close to zero. For genetic programming, it is easy to find relatively close fit solutions in the form of straight lines close to the X axis. However, it is difficult to improve on these solutions to find a really good approxima-

tion, especially if the relatively fit and simple individuals spread across the genetic programming population after a few generations.

The polynomials of increasing degree with non-zero coefficients and real roots in the $[-1, 1]$ interval constitute a class of increasing difficulty problem instances which are suitable for studying the behaviour of genetic programming, as they have the following properties:

- Minimum Required Depth. Higher degree polynomials can be approximated less easily with our function set. Each term of degree d is formed by a sequence of $term_1 \times term_2 \times \ldots term_d$. This, in general, will require larger solutions for a good approximation of a higher degree polynomial. A polynomial with real roots and non-zero coefficients $P(x) = \prod_{i=1}^{d}(x - a_i)$ can be expressed as a tree by using $2d - 1$ functions (i.e., $d - 1$ multiplications and d subtractions) and $2d$ terminals. The minimum depth (thus complete) tree representing this polynomial is of depth $2 + log(d)$.

- Maximum Allowed Depth. It follows from the above requirement that polynomials with higher degree are harder for genetic programming to approximate if allowing the same limited depth for trees, as there is less space for genetic programming to grow introns. Introns refer to non-functional code in the program. In these cases the trees must be more efficient in using more nodes to express the solution.

- Approximation Ability. Lastly, the Matlab Polyfit routine is used to assign a degree of approximation ability to the polynomials. Polyfit finds the best polynomial of given degree to fit a set of points. The polynomials of degrees 11, 7 and 3 (shown in Figure 6.1) were best fitted as polynomials of degrees 11, 8 and 3, respectively, with errors of order 10^{-15}, the approximation error being the highest for the 11 degree polynomial. That is, there were no close approximate polynomials of degree i to generated polynomials of degree j with $i << j$. Furthermore, as the generated polynomials are best fit with Polyfit polynomials of similar degree, a successful application of genetic programming on this problem should also find polynomials of similar degree.

The type of increased difficulty posed by the random polynomials and the Binomial-3 function is similar. The Binomial-3 problem must cope with ERC values spanning an increasing range. Likewise, for the random polynomials, when the degree increases, the fixed ERC range will become less appropriate as the polynomials contain variations of smaller magnitude.

For assessing the differences in behaviour between easy and difficult instances, two measures of diversity are used. The entropy measure is used for indicating the distribution of individuals over fitness values (and subsequently the level of selection pressure) and the edit distance measure is used for indicating the structural diversity of a population. These measures are described next.

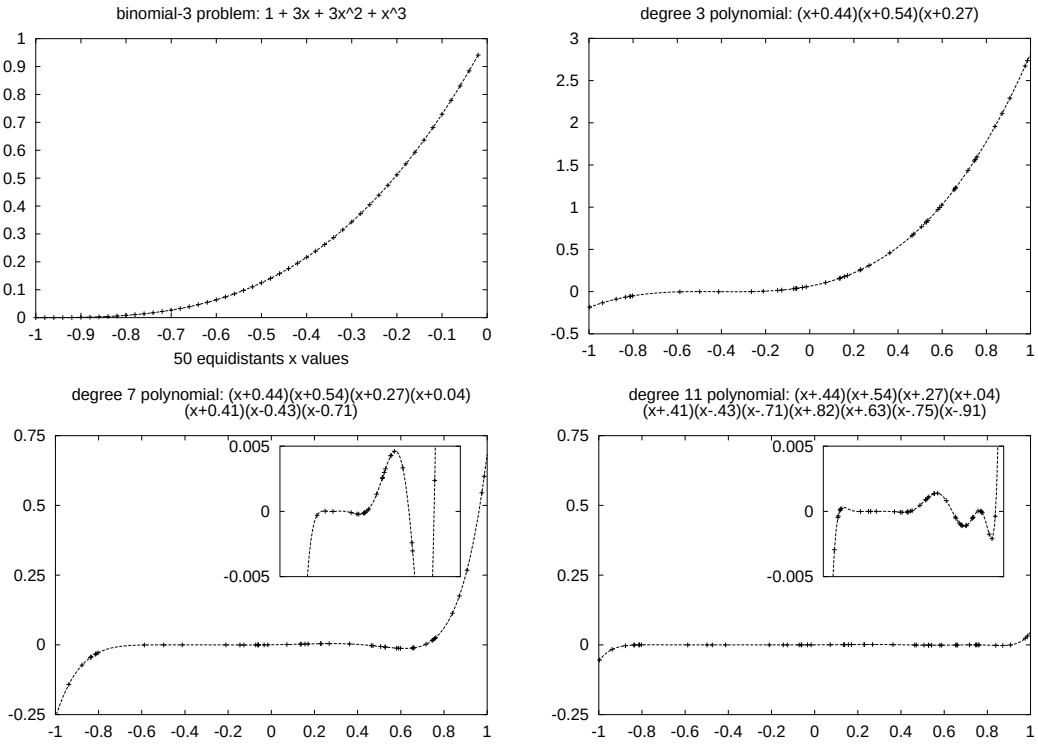


FIGURE 6.1: The Binomial-3 problem and the generated polynomial functions of degree 3 (upper right), 7 (bottom left) and 11 (bottom right). The inset for degree 7 and 11 shows the same x-range and smaller y-range between [-0.005:0.005], where it is relatively easy for genetic programming to find a *close* fit, but difficult to fine-tune the approximation.

6.2.1 Population Measures: Entropy and Edit Distance Diversity

To better understand the distribution of fitness values that a selection method is presented with, and the ability of the population to represent solutions for each problem instance, we use the measure of entropy based on fitness, described in detail in Chapter 4. Again, the entropy measure represents the chaos of the system with respect to the distribution of fitness values amongst the population (Rosca, 1995a). An increase in entropy reflects the system (population's fitness values) moving into a state of more chaos (more different fitness values). This measure also gives us a sense of how the fitness values are distributed over the fitness classes, from a uniform distribution over all fitness classes to the other extreme of the entire population belonging to only one fitness class.

Measuring the genetic diversity of populations is difficult, as there are many aspects of tree shapes and contents that could be measured. In this study, a measure is used based on the edit distance between two trees, introduced and used in Chapter 4 as edit distance One, and in Chapter 5 as the non-weighted edit distance. This measure is used to understand how structurally similar populations become and to give insight into the search process.

TABLE 6.1: Experiment and problem parameters for the Binomial-3 and random polynomial experiments.

Population size	500
Functions	+,-,*,p/
Terminals	x, ERCs
Polynomial ERC range	$[-10, 10]$
Binomial-3 ERC range	*unity* $[-1, 1)$
	ten $[-10, 10)$
	hundred $[-100, 100)$
	control [no ERCs]
Tournament selection	size 4
Maximum tree depth	10
Ramped half-n-half tree creation	minimum depth 2
	maximum depth 4
Maximum generations	101
Other parameters same as in Chapter 4	

6.3 Experimental Investigation

100 independent random runs are performed for each problem instance. Experiment and problem parameters are described in Table 6.1. The fitness for both problems are calculated by summing the squared difference for each point along the problem instance's function and the function produced by the individual. All problems used here are minimisation problems with an ideal fitness of zero. However, we report the adjusted fitness $1/(1 + \text{raw fitness})$ (with ideal value of one). For the random polynomial experiments a random seed was selected and used to generate three polynomials of degree 3, 7 and 11. The three polynomials are:

$$\begin{aligned} degree\ 3 &= (x + 0.44) \times (x + 0.54) \times (x + 0.27) \\ degree\ 7 &= (x + 0.44) \times (x + 0.54) \times (x + 0.27) \times \\ &\quad (x + 0.04) \times (x + 0.41) \times (x - 0.43) \times (x - 0.71) \\ degree\ 11 &= (x + 0.44) \times (x + 0.54) \times (x + 0.27) \times \\ &\quad (x + 0.04) \times (x + 0.41) \times (x - 0.43) \times (x - 0.71) \times \\ &\quad (x + 0.82) \times (x + 0.63) \times (x - 0.75) \times (x - 0.91) \end{aligned}$$

The polynomials are graphically presented in Figure 6.1.

6.4 Binomial-3 and Random Polynomial Results

The results are evaluated to first validate the increase of difficulty across instances. Measures of entropy, edit distance diversity and size are examined and a general hypothesis is developed to

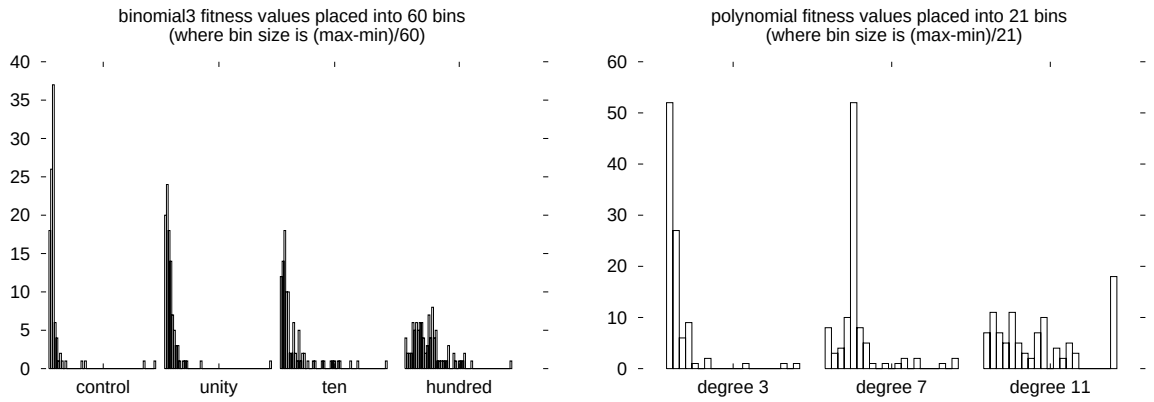


FIGURE 6.2: Each problem instance's final population performance values have been discretised and placed into 60 (Binomial-3) or 21 (polynomials) bins to show their distribution. Bins to the left in each instance represent better fitness.

explain what induces faster rates of code growth. The following section looks more closely at the correlation between growth and these measures.

6.4.1 Establishing Difficulty

Daida et al. (2001) established the tunable nature of the Binomial-3 problem by demonstrating increased difficulty through the reduced frequency which good solutions are found, the longer it took for good solutions to be found and the increased size of good solutions. In Figure 6.2, the best fitness of 100 runs for each experiment are placed into bins and plotted in a histogram. The left-most bins in each experiment – corresponding to the number of best solutions found – are expectedly larger for easier instances. The Binomial-3 results show a clear trend of less good solutions with increasing ERC ranges. In the case of the polynomial results, this trend is clear for degree 3 and degree 11. Note the large number of poor fit runs for the degree 11 polynomial (the right-most bins). These 'failed' runs tend to be present in regression problems where the lack of initial fitness improvements causes the run to become stuck in poor local optima. The nature of the degree 11 polynomial amplifies this effect, as we would expect. As shown later, the degree 7 polynomial produces a mixture of results but generally shows the expected behaviour.

In Figure 6.3 the results from Daida et al. (2001) are reproduced based on the experiments carried out here. Figure 6.4 shows the similar results for the generated polynomials. The same trends apply for both the Binomial-3 instances and the random polynomials. Increasing the ERC range worsens the best fitness found and increases the size and depth of the solutions when best fitness is found. When the difficulty is increased, the best-of-run solutions are found in later generations. Note that while the fitness function remains the same in the Binomial-3 problems, the polynomial

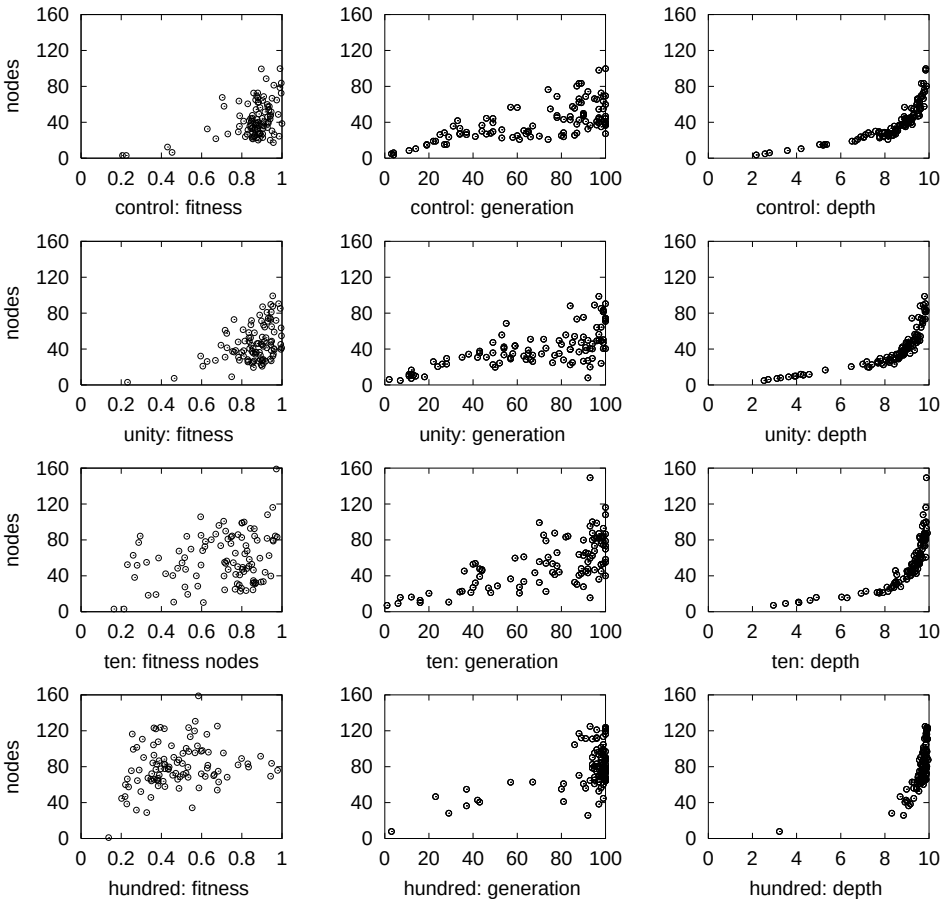


FIGURE 6.3: Results of the Binomial-3 experiments. The average size of the population containing the best-of-run individual is plotted against best-of-run fitness (left column), the generation in which the best-of-run individual occurred (middle column) and the average depth of the population (right column). Each circle corresponds to one run.

fitness function does change. For the polynomial experiments, the distinguishing feature of run performance is the amount of improvement that genetic programming can find. The difference in this behaviour can be seen in Figure 6.6.

The higher degree polynomials contain smaller variations and the fitness functions across the instances are not the same. As the higher degree polynomials start with lower initial fitness, the improvement rate from initial generations to later ones is observed as a sign of performance. The higher the degree, the less improvement is made during the evolutionary process, the bigger and deeper solutions are and the later they are found. A more in-depth analysis of the behaviour of runs with respect to size, entropy and edit distance follows.

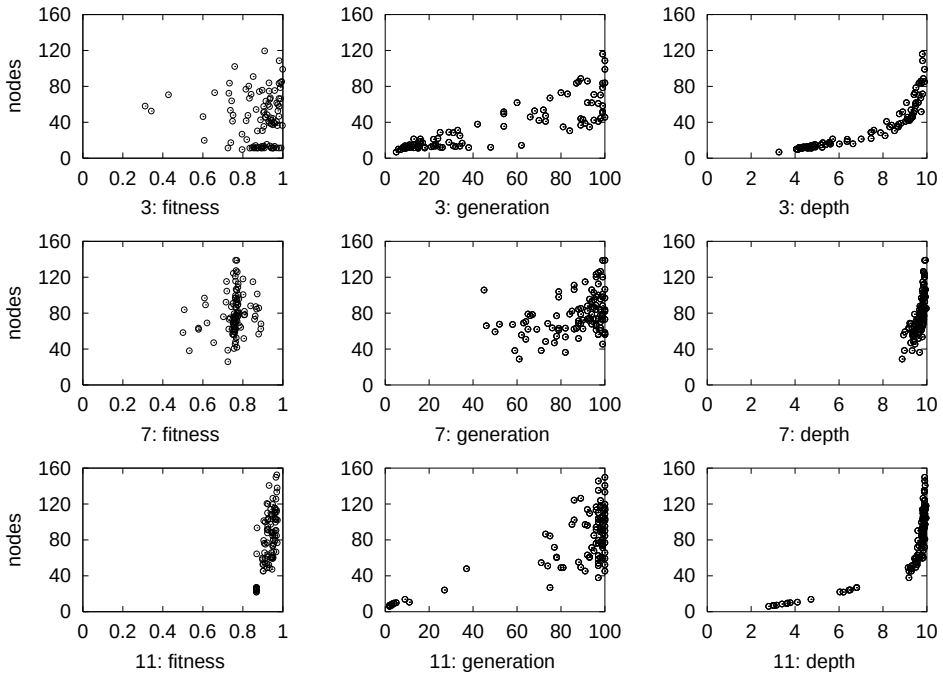


FIGURE 6.4: Results of the random polynomial experiments. The average size of the population containing the best-of-run individual is plotted against the best-of-run fitness (left column), the generation in which the best-of-run individual occurred (middle column) and the average depth of the population (right column). Each circle corresponds to one run.

6.4.2 Binomial-3 Results

For the increasing range of ERCs, the instances (*control*, *unity*, *ten* and *hundred*) become increasingly difficult for genetic programming to solve. As shown in Figure 6.5, fitness (adjusted fitness) converges less quickly, fewer good fitness values are found and individuals become larger and deeper for the larger ERC ranges. Additionally, the edit distance diversity is slightly higher for the easier instances. This would seem counterintuitive as one would expect earlier convergence toward similar programs for easier instances. In the bottom graph of Figure 6.5, the entropy quickly falls and rises, then continues to decrease. This confirms that once good solutions are found, the population loses unique fitness values. As more and more solutions have the same fitness value, entropy decreases. However, the easier instances, in contrast to the difficult instances, are also more likely to be solved by more *different* programs. This explains why diversity is higher in this case. While the populations do converge and lose diversity, the easier instances converge to a more varied selection of fit individuals. Note that the time when entropy begins to slowly decrease also marks the time when code growth begins to slow down. For the easier instances, this occurs just before generation 40, and for the *hundred* instance, somewhere between generation 60 and 80.

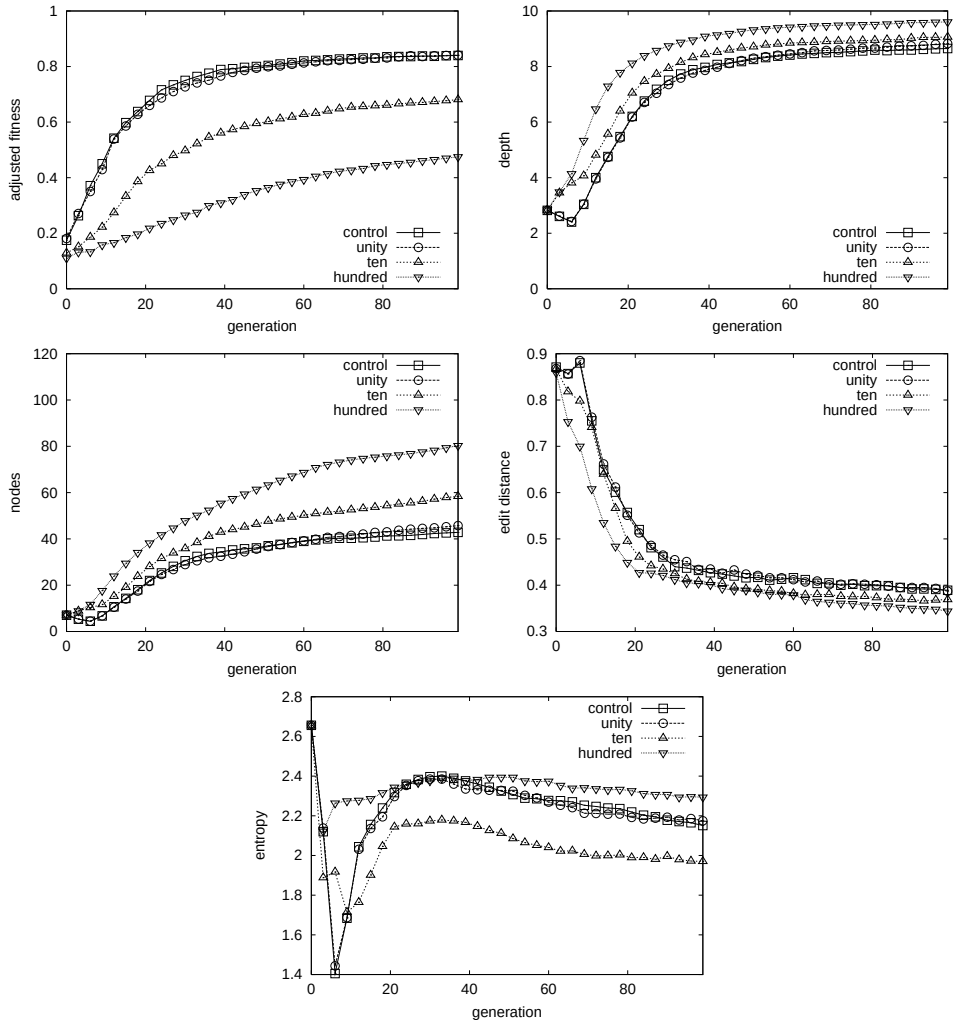


FIGURE 6.5: The Binomial-3 problem adjusted fitness, depth, nodes, edit distance and entropy versus generation for the *control*, *unity*, *ten*, and *hundred* experiments. Points represent the average over 100 runs.

6.4.3 Random Polynomials

It can be seen in Figure 6.6 that increasing the degree of the random polynomials results in a behaviour that is characteristic of increased difficulty. If one looks at the *rate* of improvement, it is decreasing as the degree increases. The higher degree instances cause a higher rate of code growth and populations with deeper trees. Again, the easier instance (degree 3) has higher edit distance diversity than the harder instance (degree 11). As noted before, the degree 7 polynomial tends to perform with less uniformity than the Binomial-3 instances. A higher rate of code growth and a lower edit distance diversity are seen in the degree 7 polynomial results.

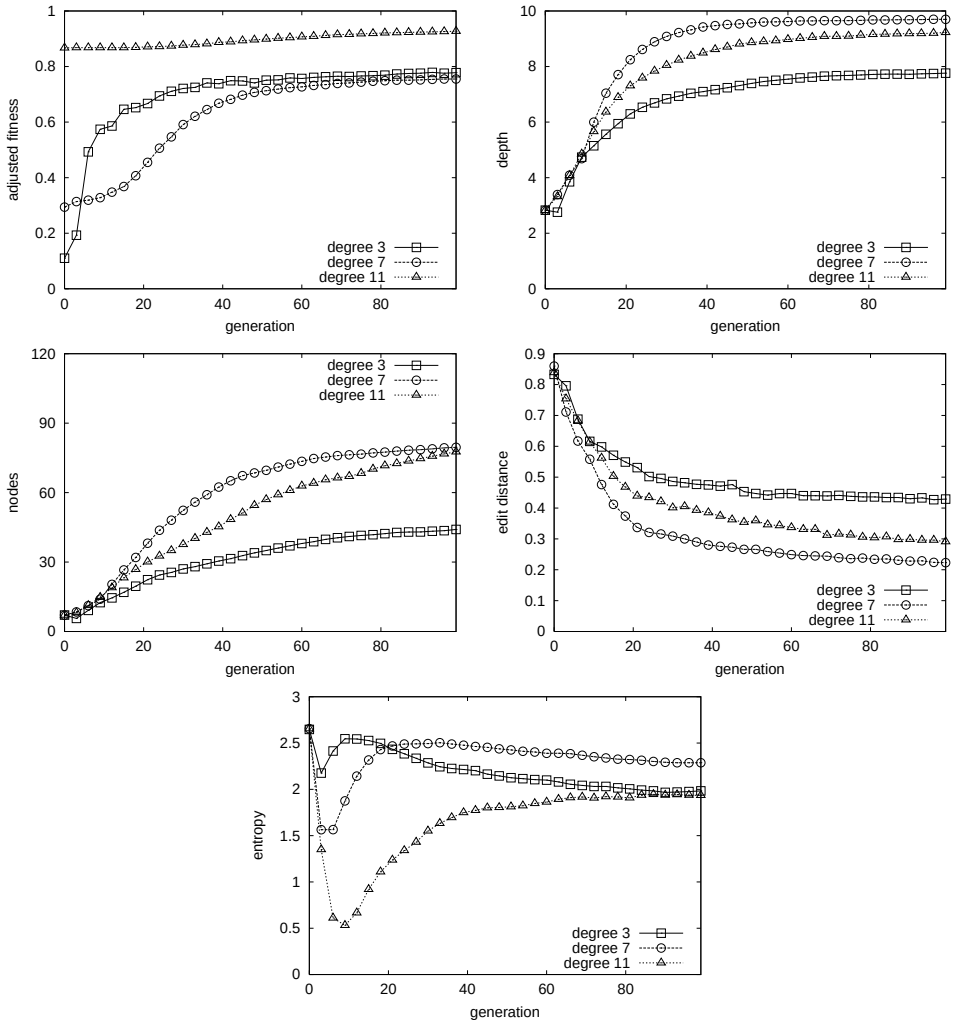


FIGURE 6.6: The random polynomial experiments adjusted fitness, depth, nodes, edit distance and entropy versus generation for degree 3, 7 and 11 polynomials. Points represent the average over 100 runs.

There is one major difference between the results of the Binomial-3 and the polynomial experiments: the initial decrease in entropy of the easier and harder instances. While the initial fluctuation of entropy is not very important for the final analysis, it demonstrates an interesting difference between the problems. In Figure 6.5, an increase in instance difficulty results in less of an initial decrease in entropy. However, in Figure 6.6 a smaller decrease in initial entropy can be observed for the lower degree polynomial. As the ERC range is increased in the Binomial-3 problem, it is likely that initial populations will be able to represent more unique fitness values than with smaller ERC ranges. While each ERC range does contain the same number of usable constants, some of these values become functionally identical, especially small numbers that may cause division by zero or be similar because of precision representation. In the polynomial experiments, the initial

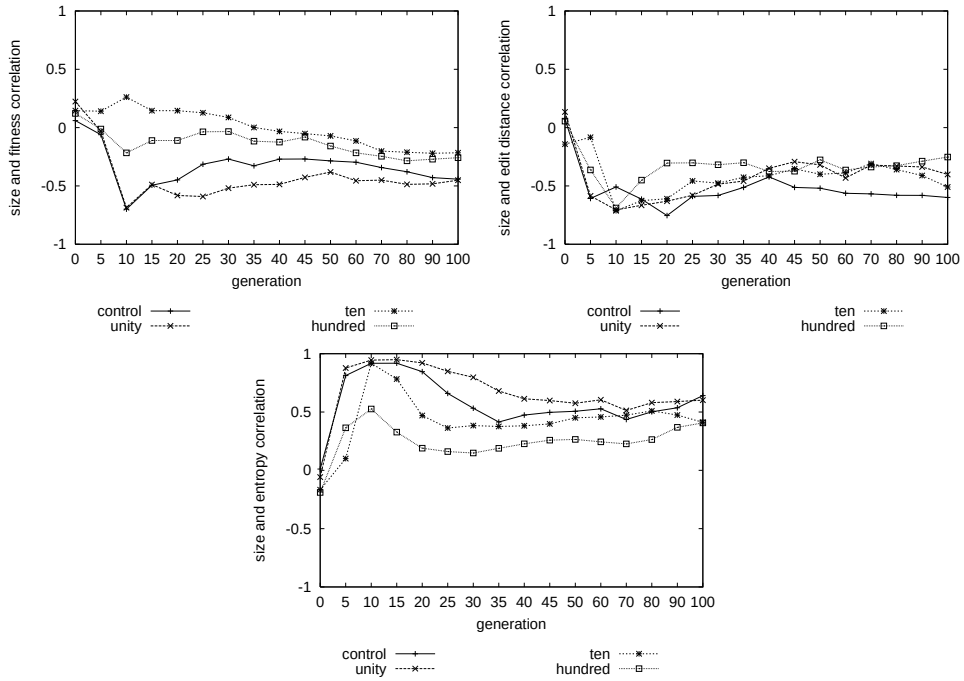


FIGURE 6.7: Spearman correlation between size and fitness and between edit distance and entropy for the Binomial-3 experiments.

greater loss of entropy by the degree 11 polynomial shows the initial difficulty genetic programming has in representing different initial solutions when the function closely represents a straight line. These observations refer to the populations that undergo the initial rounds of selection and that have higher or lower decrease in entropy.

6.5 Discussion of a Causal Model

The increase in size, depth and time needed for best-of-run solutions with increased difficulty is clear for both the Binomial-3 and polynomial problems. Also, for both problems, many small good solutions are found in early and late generations. Increased size is expected in later generations, but runs appear to have very different rates of growth. However, what is the cause of the varied rates of code growth?

By observing the behaviour of entropy in Figures 6.5 and 6.6, it can be seen that the harder instances contain longer periods of higher (Binomial-3) or increasing (polynomials) entropy. As entropy describes the number of different fitness values and their distribution in the population, lower entropy will cause selection to become more random. When selection is faced with a population of identical fitness values (the extreme of low entropy), selection becomes purely random. Thus,

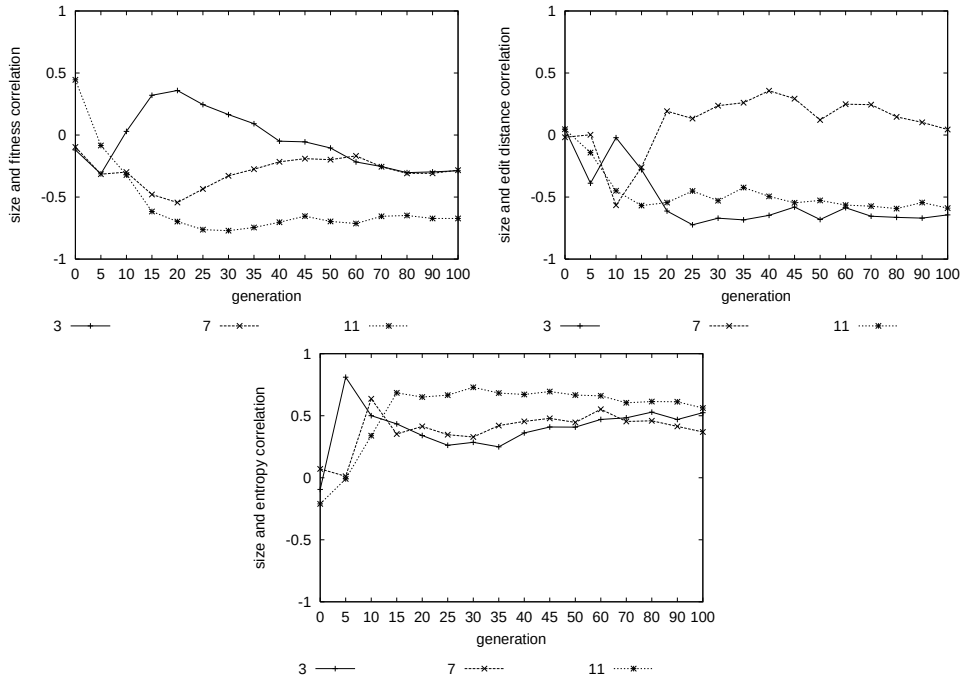


FIGURE 6.8: Spearman correlation between size and fitness and between edit distance and entropy for the random polynomial experiments.

lower or decreasing entropy in the easier instances will induce a lower selection pressure.

Figures 6.7 and 6.8 show the evolution of the Spearman correlation coefficient between the average size of a population and its best fitness (raw fitness, where lower is better), entropy and diversity, respectively. The correlation is calculated in every 5th generation up to generation 50, then every 10th afterward. Generally, size is negatively correlated with fitness (low fitness with large size), negatively correlated with edit distance diversity (low edit distance with large size) and positively correlated with entropy (high entropy with large size). There is a clear trend of bigger individuals in populations with higher entropy and lower edit distance.

The 7-degree polynomial is the exception with erratic correlation between edit distance and size, and appears to contain aspects of both the 3-degree polynomial and the 11-degree polynomial. For the 7-degree polynomial there are very few good individuals initially, and they are subsequently over-selected (similarly to the degree-11). After the initial period of code-growth, optimal solutions are represented easily by many different programs, leading to reduced entropy and code growth (as with degree-3), but lower diversity following initial over-selection.

6.5.1 Hypothesis

The increase in instance difficulty makes it more difficult to find good programs, and, when the population converges, to find one or more similarly good (optimal or sub-optimal) programs as well. In the case of easier instances, entropy decreases when the population converges toward individuals with the same optimal (or sub-optimal) fitness. In harder instances, the lack of this behaviour causes higher entropy, which does not decrease as quickly, maintaining higher selection pressure. This non-decreased selection pressure is hypothesised as a cause of more code growth and less diversity, which also causes more code growth. This hypothesis of the causal relationship between difficulty, entropy, diversity and growth is shown in Figure 6.9. The experiments show evidence of higher entropy, less genetic diversity and a higher rate of code growth for harder instances. Supporting evidence for the conjectures that higher selection pressure and lower diversity cause an increased rate of code growth is now given.

Does high selection pressure cause growth?

Tackett suggested that the rate of growth is proportionate to the amount of selection pressure (Tackett, 1994) and Langdon and Poli (1998a) showed that removing selection pressure stops code growth. To see the effect of varied selection pressure on code growth, one can look at different selection schemes and objective functions used in the literature. Smith and Harries (1998) investigated selection schemes and variable tournament sizes to show the same trend: less selection pressure generally leads to reduced code growth. Poli (2003) studied a method for reducing code growth by periodically worsening the fitness of above average sized individuals. In the control experiment with no code growth control, the results (Fig. 1, page 211) show trends of smaller tournament sizes producing less growth.

Considering the theories of code growth, particularly that individuals with similar size and shape are less likely to produce children with worse fitness than two dissimilar individuals, an increased rate of growth is expected with higher selection pressure. In this case, higher selection pressure becomes more likely to pick the same individual repeatedly. Given that the algorithm contains some bias for producing bigger individuals (as bigger children are more likely to have a higher chance of survival), the repeated selection of these individuals will consistently produce offspring which in turn have a better chance of survival.

Does low diversity cause growth?

As the population becomes more similar in shape and content, selection will mate more similar individuals. From the above argument, whether these individuals happen to be the fit or not, a

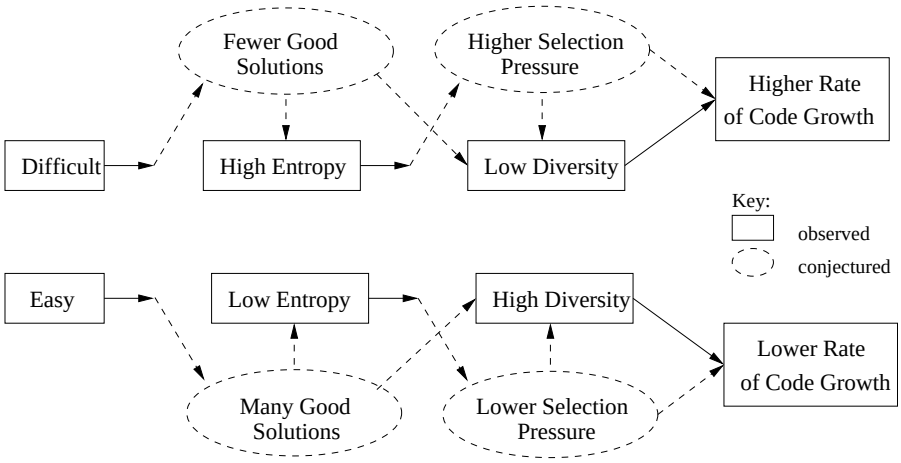


FIGURE 6.9: The symmetrical hypothesis that difficulty effects the rate of code growth by maintaining higher levels of entropy, which causes less diversity by the over-selection of better individuals, and both cause more similar individuals, which are likely to be big and to create bigger offspring.

higher rate of growth will generally occur when the individuals are of the same size and shape. When the individuals are similar in both size and shape, they will exchange similar size subtrees to produce their offspring. According to the theories of code growth, the offspring which are slightly larger will be biased to have a higher chance of survival. Also, both offspring will tend to be similar in size to their parents, slightly bigger and slightly smaller, producing more similar size individuals in the next generation and a higher rate of growth. This is in contrast to a population with high shape and content diversity.

In a diverse population (in terms of size, shape and contents), the offspring are more likely to be of varied size, i.e. smaller and larger than parents. Growth will be slower in this case. Although two dissimilar individuals may produce one larger offspring than if they were similar, these two offspring are likely to have a lower chance of survival than two offspring from similar individuals. Thus, a diverse population would not produce code growth as consistently as a low diversity population. A higher selection pressure could reverse the effect of low diversity by over-selecting the best individuals, which are likely to be similar.

Assuming code growth generally occurs with fitness improvement, does the difference in difficulty sufficiently explain the difference in entropy, diversity and rate of growth? The regression problems contain possible conflicts between content and context, noise introduced by the protected division operator and a wide range of intron and neutral code effects that could lead to varied rates of growth. Does the causal hypothesis, described in Figure 6.9, capture the important aspects that lead to an increased rate of code growth and are not influenced by these factors?

6.5.2 A Model of Difficulty

To assess the results in a model without biases introduced by a specific problem domain and fitness function, the following model is created:

1. Trees contain only 'dummy' functions and terminals which are not used to calculate fitness.
2. Fitness is assigned arbitrarily of tree shape or content. All individuals have an initial fitness of 1.0 (0.0 is the best fitness).
3. An offspring which is the same size or larger than its parent is awarded a small fitness improvement, an offspring which is smaller than its parent is penalised by a small amount. This incorporates the theories and corresponding empirical evidence of code growth.

The model consists of binary trees that are initially full with 7 nodes. The algorithm is presented below (a population size of 100, tournament selection size of 3, and subtree crossover is used for recombination):

```
Generate initial population and assign uniform fitness of 1.0
for Each generation do
  for Each individual do
    Assign with probability P the best fitness of 0.0
  for The size of the populations do
    Use tournament selection to find two parents
    Create a single offspring
    Assign the offspring the fitness of the root-parent
    if The offspring is smaller than the root-parent then
      Add 0.05 to the fitness
    else
      Subtract 0.05 from the fitness, with a minimum of 0.0
```

The P value indicates the hardness of the problem. A smaller value denotes the unlikely event that good solutions are found often and by many individuals (a difficult problem). The higher probability corresponds to the situation where many early individuals can easily represent good solutions, and in each generation more new individuals with the same good fitness can be found (an easy problem). The fact that same size or larger individuals are rewarded represents the general consensus in the community that growth is seen with improved performance in the canonical algorithm. Note that it is the *rate of growth* which is investigated here and that the assignment of ideal fitness is independent of size.

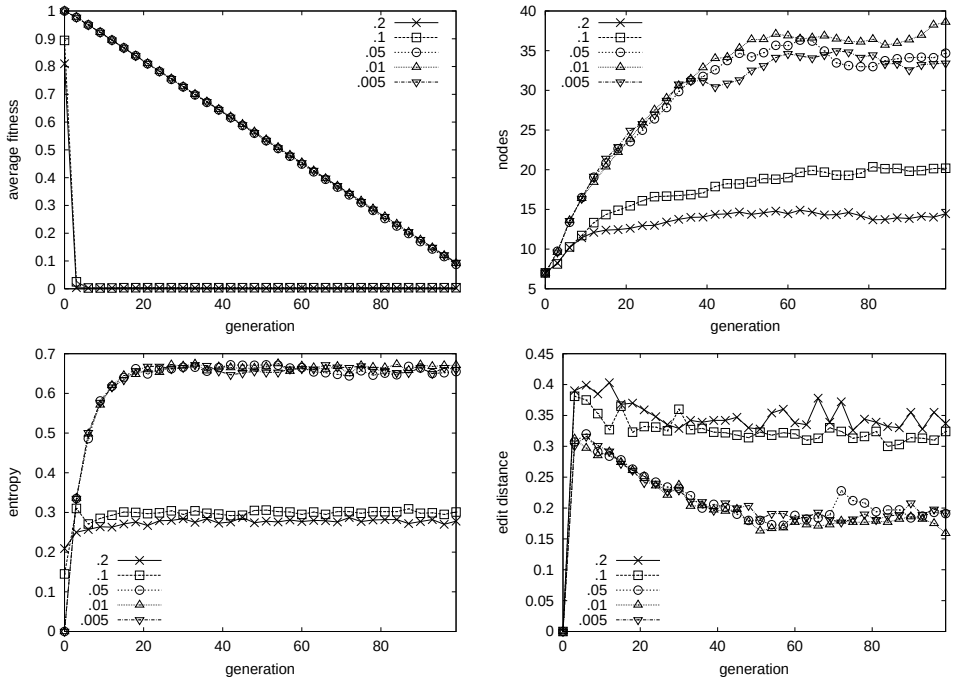


FIGURE 6.10: The model of difficulty and growth experiment results, with values of the probability term P as .2, .1, .05, .01 and .005.

A range of P values show a marked difference between easier and harder instances in Figure 6.10. Easier instances (P values of 0.2 and 0.1) induce a lower entropy and more diversity, which both contribute to a slower rate of code growth. This effect is similar to the results on symbolic regression instances presented earlier. More difficult instances induce higher entropy and lower diversity that dually contribute toward faster code growth. These results support the previous conjecture that reducing the difficulty of instances in this way leads to lower selection pressure (via entropy) and more diversity (due to less selection pressure). These factors cause more different programs to be recombined to produce less code growth.

6.5.3 Recommendations

The relationship between size and fitness is complex. Previous research has investigated the external dependencies with nodes and the node-to-node interactions in regression problems. Shifting nodes between programs is likely to result in semantic changes during crossover. An increase in program size may help to buffer against detrimental semantic changes. However, where size is not a necessary part of the fitness function, an increase in size results in a computational cost which becomes an ever-increasing burden on simulation time. Additionally, the many attempts to reduce size that result in weaker fitness illustrate the negative consequences of directly pressuring size.

Given that genetic programming is likely to favour larger trees, several methods have been used in an to attempt to limit code growth. The standard method to deal with code growth is to place a limit on the size of individuals (Koza, 1992; Luke and Panait, 2002a; Poli, 2003). This measure has been shown to be difficult to beat with respect to overall algorithm performance. However, ideal sizes for particular problems are generally unknown, and placing a limit on size can have unforeseen consequences as trees will find it harder to replicate within size limits. Also, when genetic programming is applied on more complex problems it is not clear how to scale the limits of size appropriately.

Parsimony pressure is a common method used to control size (Burke et al., 1998; Iba et al., 1994; Luke and Panait, 2002a; Luke and Panait, 2002b; Rosca, 1997a). This method favours fitter and smaller individuals. The degradation in performance due to parsimony pressure is studied by Soule and Foster (Soule and Foster, 1998). They noted that pressure to reduce size increases the failure of runs, where populations often converge to very small, poorly fit individuals. Several variations of such methods exist which vary the relative importance of size and fitness. Their drawback is that they require a large parameter search for optimality (Luke and Panait, 2002a). In (Zhang and Mühlenbein, 1995), an adaptive pressure allowed an initial increase in size before increasing the parsimony pressure. This allowed the method to find better fitness but required problem specific tuning of parameters. Other methods add a size objective to the fitness objective, creating a multiobjective problem (de Jong et al., 2001; Ekárt and Németh, 2001). Smaller size individuals and similar fitness with smaller population sizes are reported benefits of these methods. Using specific recombination operators and methods which combine like-sized individuals (Langdon, 2000b) or requiring an improvement in fitness in children (O'Reilly and Oppacher, 1995) are other approaches to reducing code growth. These methods attempt to reduce the amount of ineffective code. Generally, without a large parameter search or problem specific modifications, these methods are likely to reduce code growth but also worsen fitness.

However, high rates of growth do not seem to be necessary for solving more difficult instances, as many small good solutions are still found. Rather, an increased rate of growth appears to be the effect of higher selection pressure and lower genetic diversity. Methods which remove large individuals or pressure the population away from increased growth directly are likely to have negative effects. In these cases, good fitness is likely occurring in the neighbourhood of large individuals, and by biasing the search away from them, overall performance could be worsened. Therefore, the following is recommended as a method to reduce the rate of growth and the overall size of solutions.

Methods which work to reduce the rate of code growth more gradually or indirectly, such as the adaptive control of selection pressure and population diversity during recombination, may be a more effective strategy against code growth.

- The decrease of entropy, the poor ability of finding good initial and dissimilar solutions and the low dissimilarity between parents during recombination are possible signs that a higher rate of code growth will follow.
- When these events occur, decreasing selection pressure and encouraging the recombination of more dissimilar individuals are possible ways to lower the rate of growth while maintaining good search.

6.6 Summary

This chapter investigated the effects of population diversity by examining the relationship between problem difficulty and varied rates of code growth. Two problems with different types of difficulty and a simplified model of genetic programming have been used to understand this relationship. A causal hypothesis is formed that links difficulty to higher entropy, which in turn causes more selection pressure and loss of diversity. Increased selection pressure and lower diversity are then responsible for a higher rate of code-growth. This hypothesis is consistent with previous research. A recommendation for controlling genetic diversity, entropy and selection pressure is made to gradually effect the rate of code growth, in response to more direct methods which often have negative effects.

The results in this chapter were based on experiments in the regression domain, using a variety of problem instances. Earlier chapters displayed varying levels of results between problem domains, but diversity and fitness trends were generally consistent. Thus, the conclusions and recommendations from this chapter are relevant for other domains as well. The correlation between fitness and diversity was investigated in Chapter 4, which was complemented in this chapter with the investigation of the relationship between size and fitness, size and edit distance diversity, and size and entropy. Lastly, while the causal hypothesis of code growth appears consistent with results presented in earlier chapters and in the literature, additional research would be beneficial to confirm specific aspects of the hypothesis as well as its validity on other problem domains.

To further understand the role of population diversity in genetic programming, the next chapter examines the ability of dissimilar individuals to produce good offspring.

CHAPTER 7

DIVERSITY, SURVIVABILITY AND A NICHE FOR ISLAND MODELS

Population diversity effects many aspects of the evolutionary process. For example, the number of unique fitness values in the current population effects selection pressure. Also, a high amount of genetic differences, indicated in previous chapters by high edit distance diversity, is likely to produce slower code growth. Low amounts of genetic differences, or low edit distance diversity, describes convergence and can lead to lower selection pressure. Diversity is obviously an important topic in genetic programming. Without adequate diversity the population may be unable to produce variations to improve solution quality. However, increasing diversity can prevent sufficient exploitation and convergence. With respect to search, how do *diverse* individuals contribute to the evolutionary process? That is, how successful are the members of the population, which account for the most genetic dissimilarity, in producing new solutions during search?

The survivability of dissimilar and fit individuals is examined in this chapter to establish their role in the search process. The Tree-String problem is introduced and used in an empirical study. This problem has many similar attributes to other domains and constructed problems. The results from this study are compared with a survey of distributed and related methods that use diverse and fit individuals differently. Chapter 4 surveyed previous measures and methods of diversity. In this chapter, that survey is complemented with a survey of distributed models and related methods. Based on the analysis of the following study, a detailed proposal is made to improve genetic programming search by encouraging the recombination of genetically similar individuals by means of a speciation island model.

7.1 Previous Distributed Evolution Work

Many methods have been proposed and used in evolutionary computation to control diversity, prevent convergence and to distribute individuals over different areas of the search space. These include niching methods, diversity methods, mate selection techniques and distributed population models.

The island model (Cohoon et al., 1987) is an example of a distributed population model where subpopulations are isolated during selection, breeding and evaluation. Islands focus the evolutionary process within subpopulations before migrating individuals to other islands. There are many variations of distributed models, e.g. islands, demes, and niching methods. However, by tuning the parameter settings of each model, the functionality of different models can become very similar. So, the different parameter and algorithm settings largely define the objectives of these models with respect to similarity and dissimilarity mating, elitism and selection pressure.

Other methods simulate distributed evolution by using specific forms of selection and recombination. For example, the method of lineage selection in Chapter 5 could be implemented to only allow the recombination of individuals from the same lineage, which are very likely to have similar shape and content. In effect, lineage selection would encourage a kind of similarity-based mate selection. Previous methods that focus selection and recombination on particular individuals, or on particular features of individuals, are also included in this section. Initially, methods from evolutionary algorithms are examined, followed by applications specific to genetic programming and the tree representation.

7.1.1 Evolutionary and Genetic Algorithm Models

The most common form of island model uses fitness-based probabilistic selection for migration selection and replacement. Breeding and evaluation are typically carried out in isolation on each island. Pettey (1987) designed a distributed model based on the polytypic concept of a species being represented by several types that are capable of mating and producing viable offspring. Every generation, migration sent the best individuals in each population to each neighbour, replacing the worst individuals. Tanese (1987,1989) presented a parallel genetic algorithm implemented on a hypercube structure. Migration occurred periodically, where migrants where selected according to fitness and replaced individuals selected based on fitness in the receiving population. Belding (1995) extended the work of Tanese (1989) where migrants were selected by choosing the first n individuals in the local population according to a predefined ordering, effectively simulating a more random migrant selection strategy.

Whitley et al. (1997) investigated whether the island model has a natural advantage for finding so-

lutions to linearly separable problems. The authors concluded that for problems where an increase in the population size does not yield better results, island models may better search more of the solution space. Also, Whitley et al. noted that migration can be an "additional selective pressure", which could make the island model ideal when adding this pressure is beneficial to the problem. Cantú-Paz (1999) used the *building block* schema theory to develop equations to predict deme sizes and topologies for the parallel island model in genetic algorithms. The author concluded that the specific communication topology is probably irrelevant given the degree of connectivity of the topology.

The concept of random migrant selection and replacement was introduced in the first *island* model for evolutionary algorithms by Cohoon et al. (1987). This island model was based on isolated populations in different geographical locations. Populations had different environments by defining fitness relative to the population and individuals migrated according to a mesh and hyper-cube topology. To be consistent with the biological motivations, it was noted that migration should occur after a period of stasis. However, due to the difficulty of defining stasis or equilibrium, migration occurred after G generations, or an epoch. Unique to this model is that migrant selection is not done probabilistically but randomly to simulate a "catastrophe", or random environmental change.

Martin, Lienig and Cohoon (2000) described the results of using the island model on a VLSI design problem. They showed that a variable length epoch was slightly better than a fixed length. To implement a variable epoch length, the authors defined the end of the current epoch to be the point when no improvement to the most fit individual occurred for 25 generations. Results also showed that a random migration selection was more effective and that an aggressive selection strategy caused the population to get stuck in a local optima.

The idea that islands should consist of different environments was also seen in later work, often for coevolution. Potter and De Jong (1994) introduced a method for using isolated populations, or species, in a genetic algorithm for cooperation where the species are not "user-defined". The authors concluded that their algorithm works well on problems with independent variables. Potter and De Jong (2000) later presented an architecture to adaptively co-evolve components in a speciation model. Subpopulations represented separate species, which were required to contribute to the overall fitness to survive. Periods of stagnation suggested an insufficient number of species, resulting in the addition of new subpopulations. Species were tested in the cooperative problem domain and were removed if they failed to contribute sufficiently.

Lin, Punch and Goodman (1994) presented a hybrid of the island model called the *injection island genetic algorithm*. The injection architecture divided the search space into hierarchical levels. Populations that were trained on more general tasks were injected into populations with more specific tasks. Hu and Goodman (2002) described the *hierarchical fair competition* model, which defined hi-

erarchical *levels* with fitness value boundaries. Individuals undergo selection and recombination within these levels and move between them based on their fitness. Higher fitness levels undergo higher selection pressure to simulate added exploitation.

In Pei and Goodman (2001) the *cohort* algorithm (Holland, 2000) is compared with island genetic algorithms. The cohort genetic algorithm was designed to combat premature convergence, in the form of "hitchhiking" by allowing higher fit individuals to reproduce first. Offspring are assigned to cohorts based on their fitness. Adamidis and Petridis (1996) used an island model configuration for recurrent neural network training that varied control parameters in each sub population.

Another form of island models uses the concept of *demes*. These models are often implemented on grid-like topologies (stepping stone or hypercubes, for instance), where a deme is defined by the size of the neighborhood. Collins (1992) used a form of local breeding, the stepping-stone model, where individuals are placed on a grid, one per node, and can interact with neighbors in a given range. Several metrics measured the difference between breeding: allele diversity, genotype diversity, a panmictic index, and speed and robustness. Results indicated that, for the Partition problem, local mating was superior to panmictic breeding in finding both optimal solutions, maintaining allele and genotype diversity, faster optimisation, and robustness in the proportion of runs that produced an optimal solution.

Several other models have been proposed for various types of problems such as multiobjective optimisation (Zhu and Leung, 2002), multimodal optimisation (Bessaou et al., 2000) and agent-based evolutionary algorithms (Smith and Bonacina, 2003). As seen in the previous examples, island and distributed models are typically based on the goal of preserving diversity to prevent premature convergence or to allow the isolated evolution of different structures or behaviours. However, the effects of various forms of migration, selection and replacement are not always clear.

7.1.2 Genetic Programming Distributed Models

Much of the effort in distributed models in genetic programming focus on increasing efficiency or adding computational resources by means of parallelisation. Andre and Koza (1996) used a transputer architecture for the Even-5-Parity problem where each processor was responsible for the fitness evaluation and breeding of a subpopulation. This work is representative of the author's many large scale parallel implementations, e.g. (Koza et al., 1999). Tongchim and Chongstitvatana (2000) compared a parallel implementation with a coarse grain island model for a robot control problem, comparing synchronous and asynchronous versions. The authors previously showed that the standard island model gave a near linear speedup with the increase of processors, with degradation arising from communication and the synchronous issues (Tongchim and Chongstitvatana, 1999). Fernandez et al. (2000, 2001, 2003) studied various control parameters for multipopulation models.

Populations were homogeneous and migration selection and replacement were based on fitness, where migrants were copies, increasing selection pressure. Results highlighted the need of subpopulations to be have enough individuals to perform effective search.

Punch (1998) looked at conflicting reports of the effectiveness of multiple populations for genetic programming, extending earlier work by Punch et al. (1996) where multiple population on the Ant and Royal Tree problems showed little improvement. Andre and Koza (1996) previously showed super linear speedup on the Even-5-Parity problem. Problems with multiple solutions, such as the Parity problem, are thought to be good for multiple population models. However, deceptive problems are thought to be harder for multiple population models that evenly divide the total population over several processors. Punch tested this hypothesis on a sequencing problem and showed the negative effects deception can cause with multipopulation models. Stoffel and Spector (1996) showed improved results on a regression problem using a stack representation with the population evenly distributed across processors. Lastly, Iwashita and Iba (2002) proposed an island model using two types of crossover, depth dependent and standard, to simulate local and global search. New individuals in an island undergo depth-dependent crossover to prevent the destructive effects of standard crossover.

Juillé and Pollack (1996) used a parallel deme model of genetic programming. The island model consisted of a 2-D wrap-around mesh. Parents were selected locally and offspring replaced the least fit individual. Migrants were chosen from the subpopulation randomly. Poli and Page (2000) used demes to solve high-order Boolean Parity problems. To maintain diversity while increasing efficiency, the model only kept individuals with different fitnesses. An elitist approach distributed individuals with high fitness quickly among different demes. Langdon (1998a) used demes similar to Collins (1992) in a stepping stone model. The author reported that the deme implementation provided better results than panmictic genetic programming when evolving a queue and a list and on the balanced bracket problem. Tackett and Carmi (1994) studied different forms of breeding and population configurations for classification of the *donut* problem. Demes were formed by distributing the population over a grid. This model was shown to improve the generalization of solutions.

D'haeseleer and Bluming (1994) used genetic programming in an ALife scenario to study population dynamics. A tank problem, where tanks compete against each other for survival and participate in breeding and selection, allowed the authors to track genotypes and witness the emergence of demes. The authors pointed out that:

> "Whereas the concepts of natural selection and survival of the fittest are commonly referred to in evolutionary programming literature, the role of sub population isolation in species differentiation is cited less frequently. Identifying the environment of an individual, including the members of its local population, as a primary influence on the continued development of the local sub population as well as the species as a whole is an important step toward more realistic modelling of evolutionary mechanics."

Each individual produced a behavior signature (for phenotype diversity) that described their performance after competing against thirteen seed tanks. A genotypic diversity was measured by comparing the frequency of terminals and functions in individuals. The tank individuals competing against neighbors showed the emergence of demes, where a deme is a local population, by means of a correlation measure for phenotypes and genotypes. Parents and offspring were selected and inserted locally.

7.1.3 Speciation, Niching and Other Methods

The previous five models all used the concept of demes, while the following two rely more heavily on the notion of niches and do not use migration. Kishore et al. (2001) applied genetic programming to an n class classification problem with a "niching" technique and a parallel implementation. Niching allowed the populations to evolve independently of each other on different problems and in parallel. Bongard (1999) studied the effects of evolving two populations that attempt to solve a regression problem with two different Fourier functions. Improved solutions were found when populations sent individuals at the end of a run to a server. New runs were then seeded with both random individuals and individuals that were previous solutions to either problem.

Other methods have been used to allow populations to occupy separate areas of the search space and to promote diversity. Some of these methods were also discussed briefly in Chapter 4. Crowding (DeJong, 1975) in genetic algorithms requires new offspring to replace the most similar individual in a pool drawn from the population. Sharing, phenotype and genotype (Goldberg and Richardson, 1987; Deb and Goldberg, 1989), require individuals in a genetic algorithm to receive a lower fitness, determined by the size of the population that has the same fitness or are genetically similar. A mating restriction scheme was also investigated (Deb and Goldberg, 1989) to prevent dissimilar individuals from the likely destructive crossover operation in genetic algorithms. McKay (2000) applied the fitness sharing method to genetic programming and found increased diversity, error rate reductions and similar performance with smaller populations as benefits.

Sharing was also applied to genetic programming using an edit distance between trees (Ekárt and Németh, 2000). Although solution quality was not improved, sharing did find similar solutions of smaller size. Structure fitness sharing (Hu et al., 2002) was applied to genetic programming to encourage the *parameter* search (functions and terminals) over similar *structures* (trees) while not requiring expensive edit distance measures. The method adjusts an individual's fitness according to how many other individuals have the same tree structure. Another method used to simulate niching or speciation is negative correlation (Liu et al., 2000). In the context of learning ensembles of neural networks, "negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks". The resulting

populations were then divided into species, where a representative from each species was included in the ensemble. McKay and Abbass (2001a, 2001b) investigated negative correlation for genetic programming on the Multiplexer problem as a way to improve diversity and prevent premature convergence.

Edmonds (2001) applied genetic programming on the sun spot prediction problem by allowing individuals to find a problem niche. By creating a number of "models" (here model is used to specify an area of the problem to solve, or niche) and selecting individuals for fitness evaluation based on their neighborhood and proximity to the niche, Edmonds hoped to allow individuals to orient themselves near those niches which are most effective. Li et al. (2002) implemented a "species conserving" genetic algorithm and tested it on several multimodal and deceptive problems. Species were found in the panmictic population by a distance metric. Species "seeds" were individuals that were at least as fit or more fit than the rest of the species. These individuals were then "conserved", or copied, into the next generation.

Lastly, a range of operators have been proposed to preserve the context or similarity of exchanged genetic material (D'haeseleer, 1994; Poli and Langdon, 1998a; Langdon and Poli, 2002; Page et al., 1999; Poli and Langdon, 1998b; Platel et al., 2003). Like distributed models, these methods attempt to improve the ability of genetic programming to sample good regions of the search space effectively. While distributed models hope to allow semi-isolated subpopulations to better sample the search space prior to convergence, context preserving and homologous recombination operators attempt to sample the search space more effectively with a better designed operator. Both similar and dissimilar mating has been shown to improve search, often producing smaller in size solutions with similar fitness (Ekárt and Németh, 2000; Ryan, 1994). Chapter 5 results, produced using lineage selection, also showed how mating (likely) dissimilar individuals can lead to smaller solutions with some loss of solution quality. Also, while specialised operators may seem intuitively better, they are often computationally expensive and the results unclear.

7.1.4 Biological Foundations of Distributed Models

Several distributed models from the previous survey were motivated by the theory of punctuated equilibria (Eldredge and Gould, 1972) and the adaptive landscape and shifting balance theory (Wright, 1931; Wright, 1932). This work is examined next to provide a more thorough understanding of these models.

Punctuated Equilibria

The paper *Punctuated Equilibria: An Alternative to Phyletic Gradualism* (Eldredge and Gould, 1972) described a new perspective on evolution. Phyletic gradualism, a hypothesis proposing that evolution proceeds at a constant rate and new species evolve gradually from ancestral species, was the common world view of evolution and the authors contended that it was insufficient to explain the fossil record. Since 1930 the model of allopatric speciation was being considered, but not seriously as the cause of evolution. In allopatric speciation, new species evolve in geographical isolation from ancestral species. The authors suggested that this model more clearly explains why the fossil record is incomplete.

The theory of punctuated equilibria says that individuals and species are resistant to change, they have an innate property that prevents easy change. The previous belief that gene flow was sufficient to keep a group of subpopulations similar enough to remain a species is contrasted with the fact that the species and individuals were probably not kept coherent by gene flow but because they had a strong resistance to change. However, peripheral isolates, populations that are at the edge of a species range, with small population sizes and considerably different environments provide enough "distance" to allow for the hard and uncommon event of speciation. Thus, one sees long periods of stasis, or equilibria, where species do not change at all, and then small bursts of success by peripheral isolates in different environments.

Sewall Wright

In the 1931 paper *Evolution in Mendelian Populations*, Wright analysed the gene frequencies in different types of populations under environmental and genetic pressures, such as recombination, mutation, and drift. It was noted that the mean gene frequency of a large population will reach a stable equilibrium. When that population is sub divided into partly isolated groups, each group will have similar mean values, if under similar conditions. However, when the groups do not have similar means, those which are more successful will grow in size and the others will decline, thus causing the mean gene frequencies of the entire population to change. Wright concludes:

> "Finally in a large population, divided and subdivided into partially isolated local races of small size, there is a continually shifting differentiation among the latter (intensified by local differences in selection but occurring under uniform and static conditions) which inevitably brings about an indefinitely continuing, irreversible, adaptive, and much more rapid evolution of the species."

Wright follows this with *The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution*, a 1932 paper that, with the 1931 work, Provine (1986) calls "...Wright's two most seminal early papers

on evolutionary theory". The adaptive landscape was introduced in this work as possible gene combinations, where peaks represent high adaptiveness. The main obstacle for a species would be to effectively explore the different peaks of an adaptive landscape that would be constantly moving in response to environment changes. To solve this dilemma, Wright stated:

> "In order that this may occur, there must be some trial and error mechanism on a grand scale by which the species may explore the region surrounding the small portion of the field which it occupies. To evolve, the species must not be under strict control of natural selection. Is there such a trail and error mechanism?"

After showing the different ways a species may move around in the adaptive landscape, Wright described how a large species divided into small local "races" would be able to spread across the adaptive landscape. The local races would rapidly explore their area of the landscape and when one finds a higher peak than the currently known, it will move the entire species toward that peak. Wright concluded:

> "The conclusions is that subdivision of a species into local races provides the most effective mechanism for trail and error in the field of gene combinations."

7.1.5 Comments on Previous Distributed Models

The goals of distributed models and techniques such as niching is to maintain a population that occupies different areas of the search space. There is, however, no guarantee that these methods will perform as expected. Methods may fail to apply the right amount or kind of pressure to keep a population distributed or allow fitness improvement. The previous work described in Section 7.1 highlighted the variations of models for genetic algorithms and the more specific focus on efficiency in genetic programming models.

Distributed models commonly use panmictic-like subpopulations that perform the standard evolutionary algorithm within each island, migrating fit individuals between islands to replace the worse fit. Fitness-based migrant selection and insertion can be considered an added selection pressure, especially when migrants are copied. However, deme models can reduce selection pressure by only conducting selection within demes. Previous studies also highlighted the importance of connectedness of the topology, rather than the topology itself (Cantú-Paz, 1999). As diversity controlling methods are, in a sense, largely countering the effects of selection, the type of selection pressure a model defines is likely to be extremely important in determining its effectiveness.

Lastly, in distributed populations, migrants are likely to be, at least initially, genetically different from their new subpopulation. This is particularly true if islands converge toward different regions of the search space. However, previous work does not typically address the actual role of migrants.

It is questionable if migrants effectively add new genetic material, takeover the new subpopulation or are selected at all in their new subpopulation. The following study examines the role of dissimilar individuals, relevant to both standard populations and distributed models.

7.2 Survivability of the Diverse

In the context of search, the recombination of population members would hopefully lead to improved solutions. However, in genetic programming, there are arguments for both preventing *and* encouraging the recombination of dissimilar solutions. The ability of recombination to improve solutions can be studied by measuring the change in fitness between parent and offspring. As the root parent tends to contribute most of the genetic material, the root parent and offspring relationship is initially considered. However, instead of measuring the absolute value of fitness and its change between parent and offspring, the relative rank of parent and offspring in each respective population will be used to indicate which individuals are guiding the search more effectively (also used in (Luke, 2003)). That is, even if the overall offspring population's fitness is worse, it is still desirable to know which of these individuals contributed to search.

In the following empirical study, the survivability of diverse individuals are studied to understand the guiding forces in search. The phrase *diverse individuals* refers to those individuals in the population that account for the majority of dissimilarity. These individuals are found by comparing their average dissimilarity to the population's average dissimilarity. Following this study, many adjustments may be possible to improve efficiency or performance. Additionally, there is another important motivation for studying the survivability of diverse individuals in standard genetic programming populations: the role of the migrant in distributed models. Generalising these results to describe the expected behaviour of migrants in a distributed model is not a stretch of applicability. Distributed models are commonly defined in a way to mimic several single population runs with periodic exchange of individuals. The following study, which also considers the most dissimilar individuals as migrants, is probably being conservative as migrants are likely to be much more dissimilar in real distributed models. Next, the *genetic outlier* definition and the measure of *survivability* are described.

7.3 Genetic Outliers and Survivability

Genetic outliers are defined in a genetic programming population by first dividing the population into the *fit* and *un-fit*. The fit, which are more likely to be selected, are further divided into the similar and the dissimilar using a pair-wise distance metric. Figure 7.1 shows these divisions. Individuals which are dissimilar from the rest of the population are unlikely to have much of a

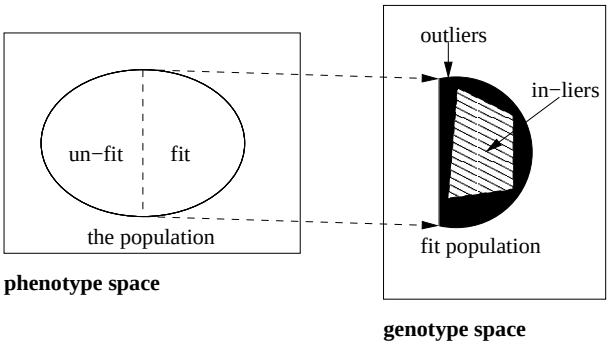


FIGURE 7.1: The phenotype space is divided into the fit and un-fit. The fit genotype space is then divided into the outliers (filled region) and the in-liers (shaded region). Of course, this example is does not represent the true relationship between phenotype and genotype spaces, it is only an example of defining the outlier individuals.

role in the evolutionary process if they have low fitness and are unlikely to be selected. Therefore, the definition of genetic outliers considers both fitness and genetic differences. Next, survivability is measured in these subpopulations by counting the number of times members are selected for recombination and the number of times their off-spring are selected in the next generation. This denotes the offspring's relative rank in the population and the original individual's survivability.

The previous chapters have shown how trends in diversity and convergence are similar between problem domains. At the same time, Chapter 5 highlighted that the type of search genetic programming carries out in each domain can vary. Specifically, domains sample structures and behaviours differently, indicating a varied response to different levels of diversity. This was also seen in Chapter 4, which showed the varied correlations between diversity and fitness on different problems. As constructed problems often focus either on the search for structure or content, but not both, to initially investigate the role of genetic diversity during the evolutionary process, the Tree-String problem is developed.

The Tree-String problem has two objectives: to match a *target tree* structure and to match a *target string* of symbols realised on the structure. This problem has explicit goals of searching for structures and contents which are likely to be conflicting, but are representative of the general class of problems genetic programming is applied to. A much smaller study will be performed later using the problem domains from previous chapters. This later study is carried out to provide additional confirmation of the applicability of analysis and to continue to improve the understanding of those domains.

7.3.1 The Tree-String Problem

The Tree-String problem consists of the following components:

- A target tree is defined with nodes labeled as leaf (l) and non-leaf (n). In this study, these structures will be binary and produced with a random tree growing method shown to be consistent with standard genetic programming (Daida, 2002). The method is also similar to those used in designing tree-based hill-climbing methods, e.g. (Juels and Wattenberg, 1995). Pairs of nodes (representing two child nodes) are repeatedly added to a tree, beginning with only a root node, until a predefined size is reached. The point of attachment of the child nodes is selected by assigning each leaf node an equal probability prior to each growth phase.
- A target string is defined consisting of symbols from the alphabet S with length equal to the number of nodes in the target tree. The target string is generated by iteratively selecting random symbols from S.
- Two objective functions are defined for the target tree and target string as:
  - Given a *breadth-first* traversal of the target and candidate trees, with nodes $\in [n, l]$, the longest common subsequence is found between the resulting strings. The target tree size minus this value is returned for the tree objective.
  - Given a *depth-first* traversal of the target and candidate trees, with nodes $\in S$, the longest common subsequence is found between these resulting strings. The target string size minus this value is returned for the string objective.
- A pareto criterion is used with the two objective functions, where the *better-than* and *equivalent-to* relationship are used for selection ($.s$ and $.t$ refer to the string objective and the tree objective, respectively), and

$$i \text{ equivalent-to } j \text{ if } (i.t > j.t \wedge i.s < j.s) \ \vee \ (i.t < j.t \wedge i.s > j.s) \ \vee \ (i.t = j.t \wedge i.s = j.s),$$

$$i \text{ better-than } j \text{ if } (i.t > j.t \wedge i.s = j.s) \ \vee \ (i.t = j.t \wedge i.s > j.s) \ \vee \ (i.t > j.t \wedge i.s > j.s).$$

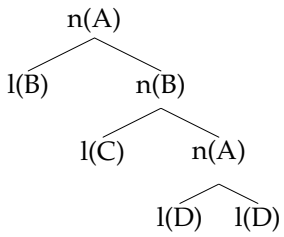
The evolutionary algorithm attempts to minimise the objective function using subtree crossover, tournament selection and other standard genetic programming parameters.

In this study, the longest common subsequence is used as a measure of similarity between the candidate and target trees and strings (Pevzner, 2000). However, other measures of similarity between trees and strings are possible, such as edit distance and sequence alignment measures from computational biology. There are other possible ways to specialise this problem for particular research agendas, where the decisions made here were designed to be consistent with common problem domains. The tunable nature of the Tree-String problem is most easily realised with the set S and the method of tree growing. In the latter case, much research has demonstrated trees that are difficult for genetic programming to reach during search (Daida et al., 2003b; Langdon and Poli, 2002). Additionally, increasing the size of the set S will increase the difficulty of minimising the string objective.

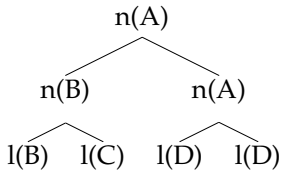
An example target string with 7 nodes is:

$$ABBCADD.$$

This string could be realised over any tree with 7 nodes. An examples of binary tree structures that are traversed breadth-first with 7 nodes is $nlnlnll$, which represents the following tree:



Another example is the tree structure $nnnllll$ that represents this tree:



7.3.2 Relevance to Other Domains

The Tree-String problem is similar to the Royal Tree problem (Punch et al., 1996) where an objective is a predefined structure. However, the Royal Tree problem gives maximum credit to full trees with specific functions and terminals at specific depths in a hierarchical fitness function. An extension of the Royal Tree (Platel et al., 2003) is also related where an objective is matching contiguous symbols in a string in the context of a linear representation. The structural aspect of the Tree-String problem is also similar to the Lid problem (Daida et al., 2003b) where a structure at a pre-defined depth and size is the objective. However, the Lid problem is concerned with structure and does not consider the effects of content or context that are present in other problem domains.

With respect to other problem domains, the following characteristics are relevant:

- The string objective is likely to contain a high level of deception, similar to the Ant problem, as described earlier and found in (Langdon and Poli, 2002). For example, the string objective function can assign the same fitness to very different strings that share the same length of matching subsequence with the target string. In the following two trees, both contain the

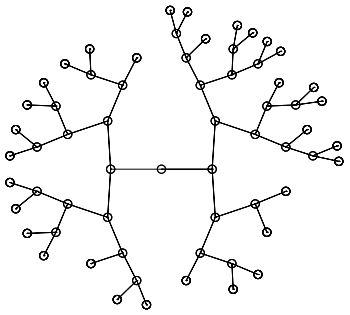
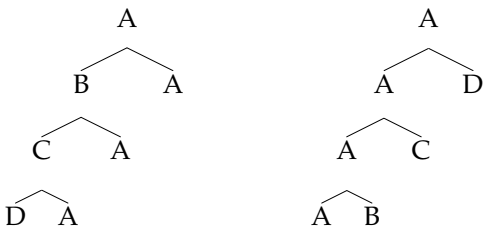


FIGURE 7.2: A binary tree of depth 10 is realised on a circular lattice on the left and the target tree in the Tree-String problem of 63 nodes on the right. The root nodes lies at the center of the lattice. Note that the scale was increased in the right figure.

string "ABC" (found by a depth-first traversal):



If the complete target string is "ABCDDDDDD", then both trees have a fitness of (9-4) by matching the subsequence "ABCD". The two trees, however, achieve the same fitness in very different ways.

- The tree objective function is likely to have less deception and be amenable to a hill-climbing search. A genetic programming approach that mimics the target tree construction method should be very effective. A hill-climbing method that encouraged the slow growth of an tree should work well in minimising the tree objective. Whether the reasons are similar, the Parity problem is also amenable to hill-climbing type methods, which often out-performs genetic programming, described in Chapter 5 and in (O'Reilly and Oppacher, 1994; Juels and Wattenberg, 1995).

- Lastly, the combined objectives of string and structure will probably create a content and context conflict similar to the regression domain (Daida et al., 2001). Moving mathematical functions expressed on subtrees is likely to change the way the content applies toward fitness in a new context. Similarly, a subtree is likely to contribute very differently when the context is not preserved in a similar shaped offspring, even one with similar content. For example, in the Tree-String problem, inserting a subtree into an existing tree is likely to effect any sequence matching the target string. This is due to the fact that the string is obtained by a depth-first traversal.

TABLE 7.1: Experiment and problem parameters for the Tree-String experiments.

Population size	500
Tree initialisation	Ramped half-n-half
Maximum generation	51
Functions (binary)	A, B, C, D
Terminals	A, B, C, D
Other parameters	same as Chapter 4 parameters

Given these properties of the Tree-String problem and the similarities to other common domains, the problem appears to be representative of the general class of problems to which genetic programming is typically applied. Of course, additional analysis is required to verify these speculated properties and behaviours of the Tree-String problem.

7.3.3 Experimental Details

A target string of length 63 is generated from $S = [A, B, C, D]$. A target tree structure is constructed with 63 nodes. Tree initialisation is bounded with minimum depth of 2 and maximum size of 4, and subtree crossover is bounded to trees with a maximum depth of 10. Experimental parameters are found in Table 7.1. The target tree is shown graphically on the lattice in Figure 7.2, where the deepest node is at depth 6. This target tree structure defines the following string (resulting from a breadth-first traversal over the nodes ('n') and leaves ('l')):

n n n n n n n n n n n n n n n n n n n l l n l n l n n n n n l n l n n l n l n l l l l l l l l l l l l l l l l
l l l l l l l l

The randomly generated target string using 63 symbols, realised over both leaves and functions, is:

C B C C C D B D A D A D B D D D D B C D C A B B D C A D C D B D C C B B B C D A
D D C B D D A B C D C A D C B A D D C A B D A

7.3.4 Genetic Outlier Definition

Figure 7.1 shows the divisions of the phenotype and genotype space used to define genetic outliers. To investigate where future populations come from, the definition of genetic outlier used here is based on the properties of selection and similarity. Selection is based on fitness, thus the population is divided into those individuals which are better-than more than half of the population. The better-than relationship describes the case where selection will always pick one individual over the other. This subpopulation is called the *fit* subpopulation. The subpopulation that is left is called the *un-fit*.

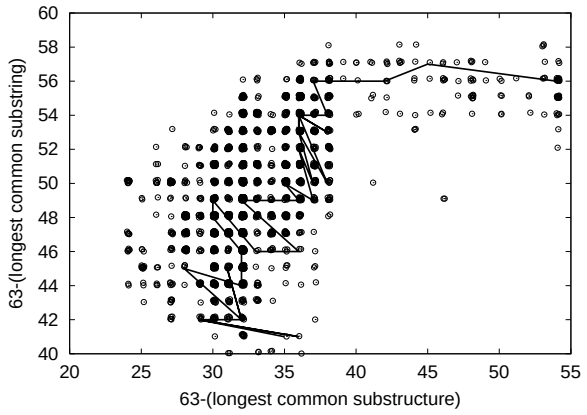


FIGURE 7.3: The pareto front and points visited for the Tree-String experiments.

Next, the fit subpopulation is further divided into the similar and the dissimilar. The Levenshtein distance between strings, or string edit distance, is used to define the distance between structures. The structure is represented by a breadth-first traversal of trees with node labels $\in$ [n,l]. This is the same distance measure used in previous chapters, called edit distance one, except trees only consist of the symbols 'n' and 'l'. Each individual's pair-wise distance is the average edit distance to the rest of the population, where each distance is normalised by dividing by the the larger size of the pair of trees. The mean pair-wise distance of the population is then found by dividing the summation of all individual pair-wise distance's by the population size. The subpopulation that is better-than half the population, the fit subpopulation, is then further divided into those which have a pair-wise distance to the population that is less than or equal to two-standard deviations from the population's mean pair-wise distance. This subpopulation is called the *in-liers*. The subpopulation that is left is called the *outliers*, which are genetically different from the rest and better-than more than half the population.

7.3.5 Genetic Properties Contributing to the Evolutionary Process

Figure 7.3 shows the fitness points visited by 40 runs of the Tree-String problem. The tree objective is along the X-Axis, and the string objective is on the Y-Axis. A randomly chosen run is highlighted by connecting the successive points this run visits. Points are randomly offset by a small amount to better illustrate their distribution. The Tree-String problem creates a complex fitness landscape where improving one objective often conflicts with the other, and vice versa.

Figure 7.4 shows the average over 10 runs of the number of outliers, in-liers and un-fit individuals (from left to right) at each generation (increasing from left to right) in the left plot. The number of selected individuals and the number of those which produced offspring that were selected in

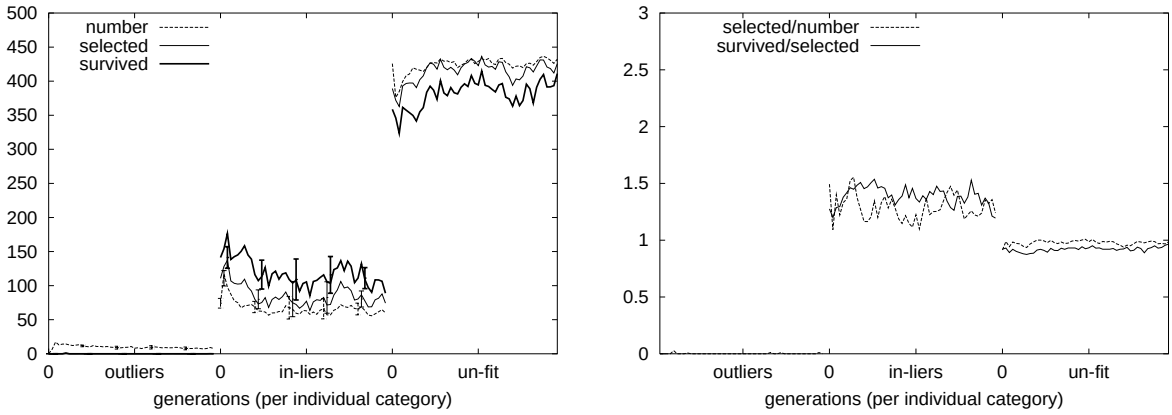


FIGURE 7.4: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Tree-String experiments. Outliers are defined by the (fitness, similarity) tuple as (better-than, 2 standard deviations).

the following generation (the survivability) are shown in heavier lines. 95% confidence bars are plotted every ten generations for the in-lier and outlier individuals. The un-fit population is the total population size minus the in-lier and outlier populations. This measure of survivability is similar to that in (Luke, 2003) where the selection of an individual represents the relative rank of that individual in the population. The measure considers the selection method being used and is a better indicator of an individual's contribution toward the search process than the change in fitness from parent to offspring.

Every generation has a number of outliers which are rarely selected, due to their low numbers and subsequently low probability of selection. Thus, these outliers have almost no survivability. The right plot in Figure 7.4 of ratios of *selected over total number* and *survived over selected* emphasise these effects. Here the in-liers generally have a higher ratio of selection, which is expected, and a higher survivability than the un-fit. The un-fit tend to have worse survivability, which emphasizes that the in-liers produce more offspring that are able to survive and contribute offspring in the next generation. The survivability of the in-liers tends to lower toward the end of the run, which is likely due to convergence, i.e. the in-liers are unable to produce useful variations.

In this initial experiment, outlier individuals have an average pair-wise edit distance greater than 2 standard deviations from the population mean and are better than over half of the population in fitness. The number of outliers in Figure 7.4 highlights the fact that the population contains good individuals that are structurally unlike the rest of the population, are unlikely to be selected due to their few numbers, and, without more experimental results, have an intuitively lower chance of producing good offspring.

The above definition of outliers addresses the concepts of fitness and genetic diversity, but it is not

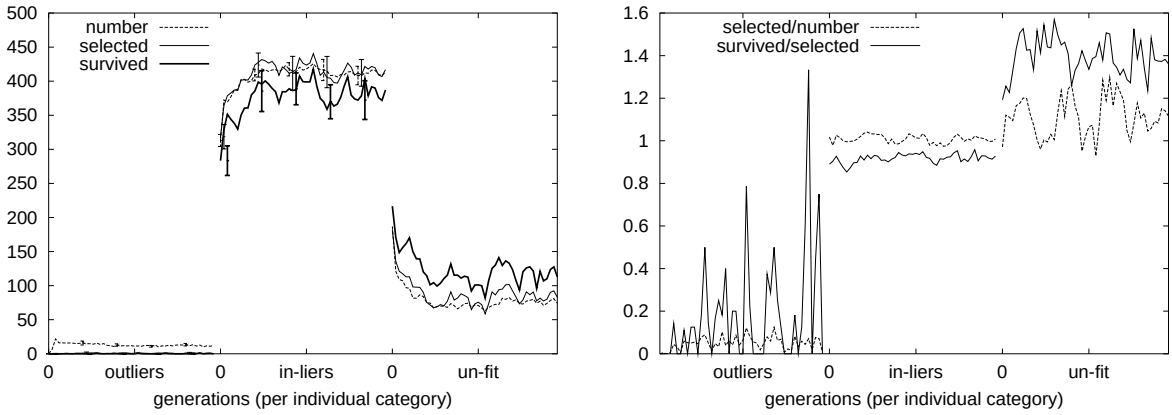


FIGURE 7.5: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Tree-String experiments using the (better-than ∨ equivalent-to , 2 standard deviation) definition of outliers.

the only possibility. For example, the fitness component in the outlier definition requires an outlier to be better-than more than half the current population. It may be beneficial to also consider individuals with equivalent fitness values in this definition. To further investigate the role of genetic diversity and survivability, three additional experiments are performed, summarised in Table 7.2, by adjusting the fitness and dissimilarity components that define outliers. The following alternative definitions provide *alternative views* of the survivability of different regions of the population. The evolutionary process is the same for all definitions of outliers, and as above, each alternative is considered using the averages of 10 random runs.

7.3.6 Alternative Definitions

Figure 7.5 shows the average and 95 % confidence bars for the random runs, but now the fitness component used to define outliers (and in-liers) is that of better-than *or* equivalent-to. With this change, the number of outliers and in-liers has increased significantly. This increase highlights the equivalence of fitness in the population. The ratios in the right plot of Figure 7.5 show dramatically different selection and survival ratios than before. The outlier survival rates are highly sporadic and seem to increase later in the evolutionary process. In the presence of many equivalent fitness

TABLE 7.2: The Tree-String outlier definition variations and respective figures.

	fitness component		difference component
Fig. 7.4	better-than	,	2 standard deviations
Fig. 7.6	better-than	,	1 standard deviations
Fig. 7.5	(better-than ∨ equivalent-to)	,	2 standard deviations
Fig. 7.7	(better-than ∨ equivalent-to)	,	1 standard deviations

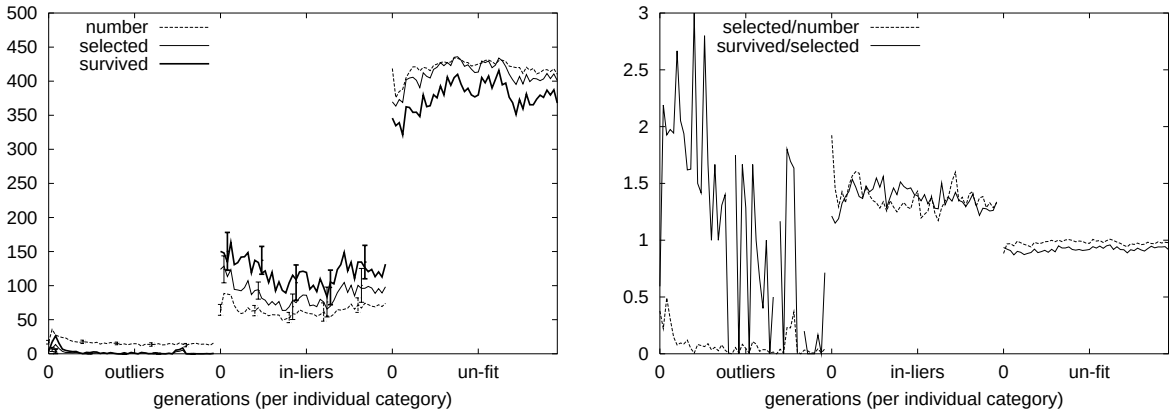


FIGURE 7.6: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Tree-String experiments using the (better-than, 1 standard deviation) definition of outliers.

relationships, diverse solutions appear to play a more important role in creating offspring which survive to create more offspring. Also, the ratios of survival are much higher in the un-fit than the in-liers. This suggests that while their overall numbers are lower, individuals that are both similarly fit and genetically similar are unable to produce viable offspring. By changing the *view* of outliers and in-liers to include equivalent fitness values in their definition, the in-lier class increased in numbers but decreased in overall survivability.

Depending on the problem representation and operator, one may wish to consider an alternative definition of genetic dissimilarity, such as the distance from the mean of only one standard deviation. Next, the the original fitness definition for outliers is used, but the genetic difference will now be based on **o**ne standard deviation. This change should allow more individuals to be considered as *dissimilar*.

Figure 7.6 shows the same plots as above for the number of outliers and the ratios of selection and survivability, but now using the one standard deviation criterion. Here, one can see again how a slight increase in the number of outliers causes their selection and survival rates to increase. The right plot of this figure emphasises the large spikes of the survival ratio of outliers. In contrast to Figure 7.5, the survivability of outliers, while still unstable, tends to decrease during the evolutionary process. The in-lier survivability frequently intersects the selected ratio, while the un-fit individuals tend to have lower survived ratios compared to the selected ratio. These results show that as the population loses diversity and becomes genetically converged, the outlier survivability becomes more unstable. When the number of outliers are high enough to be selected, they produce children with high survivability, but at an increasingly varied rate during the course of the run. This *view* of outliers, compared with Figure 7.5, shows that outliers, which have a dissimilarity between 1 and 2 standard deviations from the population's mean, play a more important role initially

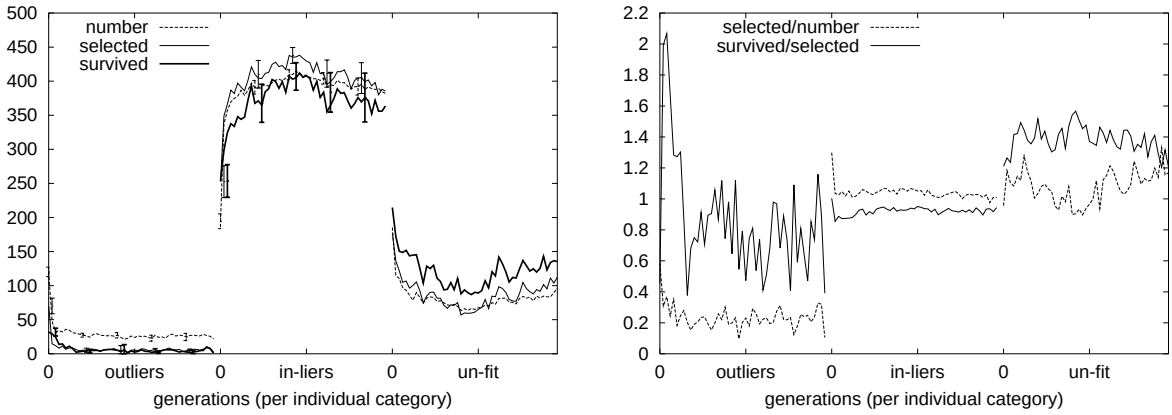


FIGURE 7.7: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Tree-String experiments using the (better-than $\vee$ equivalent-to , 1 standard deviation) definition of outliers.

in search. As Figure 7.5 shows outliers with a dissimilarity greater than 2 standard deviations, Figure 7.6 confirms that the addition of outliers with a dissimilarity greater than 1 and less than 2 standard deviations account for this increased dissimilarity. However, in the later stages of the evolutionary process, the outliers have a lower survivability.

Finally, the above two methods are combined to define an outlier as being one standard deviation from the mean average pair-wise edit distance and being better-than or equivalent-to half the population in fitness. Figure 7.7 shows that when the outlier class is made larger by including equivalent fitness and only being one standard deviation from the pair-wise distance mean, the survivability rate of outliers is consistently higher than its selection rate, but still unstable. This *view* of outliers, combined with Figures 7.4 and 7.5, shows clearly how the population converges to be mostly equivalent and genetically similar. Also, Figure 7.7 shows that when outliers are viewed using the equivalent fitness criterion, their survivability is higher for longer periods.

7.3.7 Discussion of Experimental Results

The previous experiments have provided an initial exploration into the role of diverse individuals in the population using a constructed problem. The results showed the existence of varying types of structurally diverse and fit individuals and their ability to produce successful offspring. Each of the varying definitions described in Table 7.2 shows a different perspective of the role of diversity.

The view of the population using the equivalent definition of fitness showed how the survivability of in-liers goes below their selection rates. This, combined with the previous views using the better-than relationship, shows how the increase of the number of equally fit individuals is correlated

with poorer search ability. Or, a decrease in phenotype diversity is correlated with poorer search ability. This was also seen in Chapter 4, where high fitness entropy and phenotype diversity were positively correlated with fitness improvements.

When considering the outlier dissimilarity to be one standard deviation from the population mean pair-wise distance, the low selection of these individuals still produced high rates of survival, becoming lower and more sporadic over time (Figure 7.6). What is consistent in these experiments, and with others that examine diversity with similar algorithms (McPhee and Hopper, 1999), is that the population becomes more similar in structure over time. Thus, the number of outlier individuals in all experiments tends to decrease as the population becomes more homogeneous. When that occurs, seen in Figure 7.6, the survivability of outliers becomes more sporadic. However, in Figure 7.5, using the equivalent fitness definition for outliers, the sporadic survivability of outliers increases as the population converges. With this view of outliers, they become more important as they contain diverse genetic material that the population needs to produce useful variations.

Among the possible interpretations of these results, two are initially highlighted here. First, dissimilar individuals have an unstable ability in producing offspring with high survivability. Secondly, as the population converges, and in light of equivalent fitness values, the role of dissimilar individuals becomes more important. Without considering equivalence, dissimilar individuals become less effective when the population converges. With the equivalence relationship, dissimilar individuals provide key variation necessary for variation and improvement. That is, when the space of fit individuals contain many equally fit individuals, the dissimilar individuals with the same fitness play an increasingly important role, albeit an unstable one.

Lastly, this study has grouped similar and dissimilar individuals with low fitness into the *un-fit* class. However, it is interesting to note the change of survivability rates under the varying views of the population. Most noticeably is the change between using the stricter better-than definition (Figures 7.4 and 7.6) to the better-than or equivalent definition (Figure 7.5 and 7.7). In the latter cases, the un-fit consistently produces higher survivability ratios than the outliers or the in-liers. Also, this ratio is typically higher than its own selection ratio. The loss of genetic and phenotype diversity probably increases deception and lowers selection pressure, causing search to rely on new solutions produced by the un-fit subpopulation.

An effort has been made throughout this thesis to consistently analyse genetic programming on a similar set of problem domains. In order to add to this growing body of analysis for these domains, and to provide additional validation of the above analysis using the Tree-String problem, an initial study of survivability is carried out using the Ant, Parity and regression problems.

TABLE 7.3: Experiment and problem parameters for Ant, Parity and Binomial-3 outlier experiments.

Functions	
Ant	if_food_ahead,progn2
Parity	and,or,nand,nor
Binomial3	+,-,*,p/
Terminals	
Ant	left,right,move
Parity	D1,D2,D3,D4,D5
Binomial3	x, ERCs
ERC range	$[-10, 10]$
Other parameters	same as for Tree-String experiments

7.4 The Ant, Parity and Regression Domains

The Ant, Parity and Binomial-3 regression problem instances are used with the same definitions and parameters as presented in Chapter 5. All problems are minimisation of errors; the number of missed food pellets in the Ant problem, the mean squared error in regression of the Binomial-3 problem and the number of misclassified bit-strings in the Parity problem. The genetic measure of diversity is the Levenshtein distance between tree structures denoted with [n,l]. The same experimental parameters were used as in the Tree-String experiments. The better-than relationship and 2 standard deviations are used as the fitness and difference components to define outliers.

The outliers, in-liers and un-fit distributions are shown in Figures 7.8 to 7.9. These graphs show the averages over 30 runs for each problem, where the left plot represents the numbers in each category. Survival numbers were based on the number of selected individuals that produced offspring which were selected in the next generation. 95% confidence bars are shown for the left plots and the right plots in Figures 7.8 to 7.9 show the ratios of selection and survivability.

For the Ant experiments in Figure 7.8, the rate of survival of outliers is initially high compared to their number and rate of selection, especially in the early generations. As discussed earlier in this thesis, the Ant problem contains deception and benefits from more exploration in early generations. The decrease of in-liers in the later generations is due to the increasing number of individuals which have equivalent fitness. However, even though the number and rate of selection of in-liers dramatically reduces, their offspring have a considerably higher and increasing rate of survival compared to selection.

Similar results are seen with the Parity experiments in Figure 7.9. The few number of outliers have high survival rates that are also very sporadic. In this problem, the role of in-liers and the un-fit change similarly to the Ant problem. The early stages in the evolutionary process rely on the un-fit and outliers to produce much of the surviving offspring relative to the rate of selection. However, this role shifts to the in-liers having higher survival rates in later generations. The Parity problem

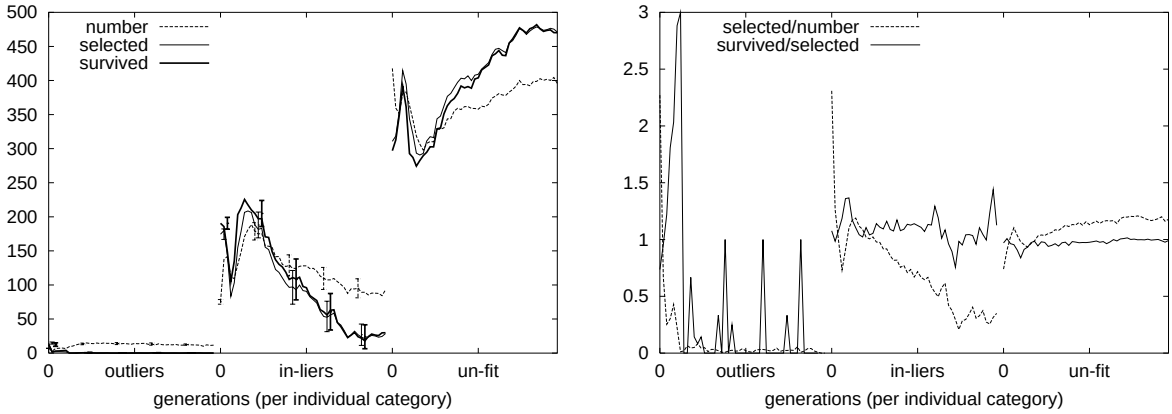


FIGURE 7.8: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Ant experiments.

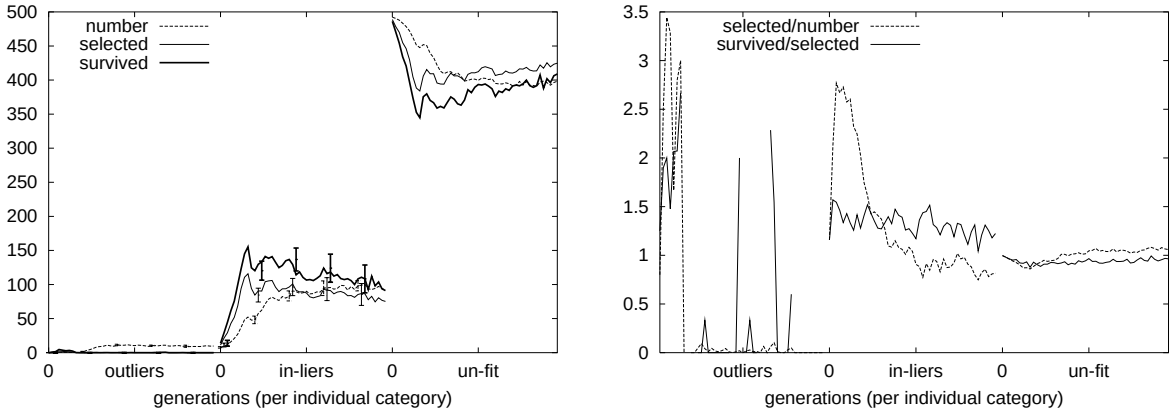


FIGURE 7.9: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Parity experiments.

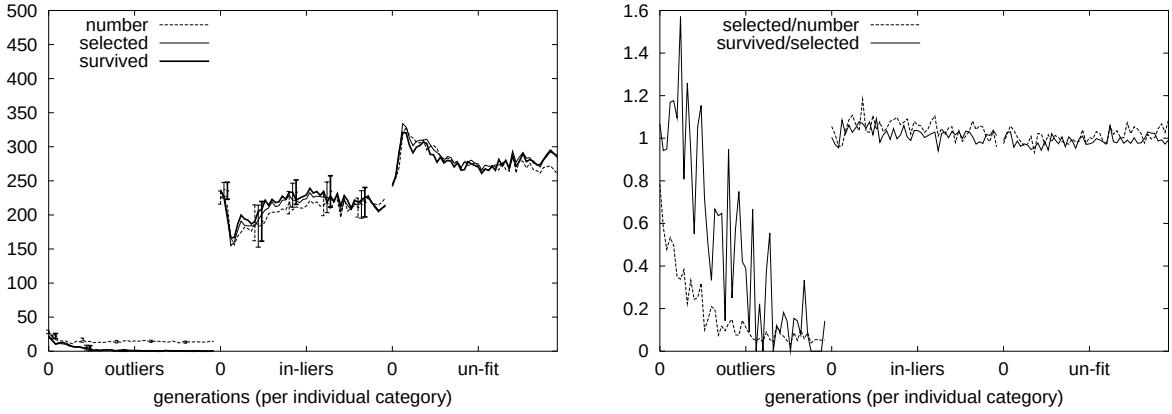


FIGURE 7.10: The average number in the population, the number of times selected and the survivability of the outliers, in-liers and un-fit for the Binomial-3 experiments.

differs from other problems (including the Tree-String problem) in that the un-fit subpopulation increases in numbers considerably. Previous chapters, particularly Chapter 5, showed the large effort genetic programming spends in sampling different solutions that have equally poor fitness. Thus, in the later stages of the evolutionary process, the un-fit subpopulation increases in numbers but becomes less effective in producing good offspring. The Ant problem has a similar behaviour, but starts the run with an already high number of un-fit.

The Binomial-3 experiments have somewhat different results, shown in Figure 7.10. These experiments contain a relatively higher numbers of outliers compared to the other problems. Outliers are selected more and achieve higher rates of survival more often. The in-liers and un-fit maintain fairly constant and similar rates of selection and survival. Outliers achieve higher rates of survival compared to their numbers and selection rates, which decreases during the run. However, it is important to remember that the definition of outliers, particularly with the difference component, is problem dependent. Performing a more extensive study of different definitions of outliers would yield more informative results, as done earlier with the Tree-String problem. Based on the above empirical study and survey of related work, a niche for islands models is proposed.

7.5 A Niche for Island Models in Genetic Programming

Previous methods used to encourage distributed evolution commonly lack explicit or adaptive methods to ensure or consider the type of genetic or fitness value distribution in the population. Many aspects of these models and methods may actually work against diversity by increasing selection and quickly migrating fit individuals. However, models like the stepping-stone model may reduce the global selection pressure by focusing tournaments only in one geographic location at a time. As previous methods do explicitly intend for populations to be distributed over the search space by using niches, islands or demes, what measures should be used to define their distribution and ensure their effectiveness? That is, given the previous experimental results in this thesis, could distributed models explicitly leverage the outliers to improve the search?

Sewall Wright's early work with the *shifting balance theory* and later work on stasis and allopatric speciation by Eldredge and Gould all rely on semi-isolated populations, fledgling populations, or peripheral isolates as the mechanism for rapid evolutionary change. Wright proposed that the semi-isolated populations that exist in separate ecosystems and undergo their own evolutionary history would have enough random evolution and genetic drift to allow for rapid evolution. Eldredge points out that behaviors like habitat tracking keep species in stasis. He says that species do not adapt first or go extinct when ecosystems change, but first try to move to another familiar ecosystem. Upon failure to do so, extinction or adaptation follow, where peripheral isolates are the candidates for rapid evolutionary change.

In evolutionary computation and distributed populations, stasis could be considered to be stability in the fitness or convergence of the genotypes. Cohoon et al. (1987) mentioned the idea of stasis in their original paper on genetic algorithms and punctuated equilibrium, but simply to say that in an ideal model a local population should reach some kind of equilibrium before migration occurs. Potter and DeJong (2000) use stasis as periods with no fitness improvement to trigger the addition of new subpopulations. Incorporating changes in the ecosystem, or catastrophes, when stasis occurs may help avoid convergence and stagnation in local optima.

However, stasis may be one possible way to allow distributed populations to emphasise the importance of outliers and dissimilar individuals. When convergence increases, the search may benefit from a change of focus from the converged population to the diverse elements. Given the results showing the sporadic survival rates of outliers, it would seem beneficial to consider the outliers in a different environment, or with special operators, to increase their survivability. Also, as the population converges and contains more equivalently fit individuals, biasing selection toward genetically or behaviourally dissimilar individuals may be another way of improving the search.

In the light of the sporadic rates of survivability, how should the existing use of migration be evaluated? Is migration necessary or beneficial? Previous methods used to preserve niches for objectives or components often prevented migration-like events. The effects of migration are likely to be determined by the degree of difference between the subpopulation and the migrant and the migrants relative fitness to the new subpopulation. In the worst case, the migrant is either never selected or repeatedly selected, quickly taking over the subpopulation with copies of itself.

Convergence to local optima and the inability to escape those optima is an important issue in genetic programming, and the island model is often cited as a solution. Cohoon et al. (1987) stated that migrant selection should be done randomly to simulate a random "catastrophe" or environmental change. However, later work typically uses probabilistic migrant selection and insertion. Instead of migration moving fit individuals to new subpopulations, perhaps it should consider diverse individuals in a new environment where they can be exploited.

Lastly, the research into the use of homologous individuals or operators in genetic programming has shown improved performance when attempting to preserve the context of exchanged genetic material and those which work on similar shaped and placed material (D'haeseleer, 1994; Platel et al., 2003; Langdon, 2000b). If a good solution is migrated to a new subpopulation, pressure from the native individuals may prevent it from evolving due to the differences between genotypes or phenotypes. However, a smaller population with less competition may allow this individual to survive. These ideas are raised in Iwashita and Iba (2002) as new migrants are protected from the likely-destructive subtree crossover by using only depth-dependent crossover on these individuals.

7.5.1 Proposed Niche Solution

Due to the previous uses of islands, niches and species described previously in this chapter, an island model is proposed. This model is based on the results from Section 7.3 and the survey from Section 7.1, and it aims to accomplish the following tasks:

1. Improve the survivability of selected individuals. By giving the operator a higher chance of success in producing better solutions, the entire search process should become more efficient and successful. Section 7.3 showed overall low and sporadic survivability rates for the fit individuals. Ideally, these individuals should be more responsible for guiding the search process.
2. Improve genetic programming's search ability by giving dissimilar individuals a higher rate of survivability. These individuals represent new regions of the search space that the algorithm would be likely to benefit from exploring.

The proposed model consists of:

- A measure of genetic dissimilarity to identify dissimilar and fit individuals that arise during the evolutionary process.
- A speciation event that creates new islands based on these dissimilar individuals.

This process would simulate a speciation event where the dissimilar individual leaves the existing population to create a new species. Speciating new islands introduces a form of branching. If the current population has too much momentum to move away from a local optima even when different, and better individuals exist, then search paths are lost. These individuals were earlier called *genetic outliers* and are defined as structurally or genetically dissimilar as well as highly fit. In addition to being dissimilar, outliers were also highly fit, giving more indication that they are promising solutions. Based on the specific representation and operator, measures of dissimilarity may also consider node content as well as structure.

Another benefit of this model is that the role of diversity becomes easier to understand and control. When a subpopulation is more similar in structure and content, the role of diversity in the population becomes clearer. If a subpopulation has converged to a single structure and has lost all diversity at the level which the operators work (near the leaves, for instance), then the search can either continue by adding diversity in these areas or it will halt. Adding diversity at the root level may provide large behavioural changes, but if the operators do not work with these nodes, typically seen with subtree crossover does not, they could be lost during selection. However, adding diversity where operators work could be a better way to continue search. Issues of diversity within

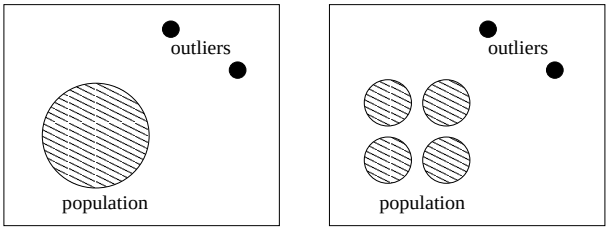


FIGURE 7.11: Two possible views of outliers in the genotype space, where the shaded regions represent the population and the outliers are represented outside by the filled region.

the canonical genetic programming search process are complex and problem dependent. This is highlighted in the previous chapters. Considering diversity issues within genetically homologous subpopulations, given a particular operator, becomes a more tractable space and issue.

In summary, the proposed niche for island models would allow a subpopulation to converge toward local optima in isolation. Migration events are replaced by a form of speciation which forks off a new search on sufficiently different individuals (structures) and further exploits that structure on a new island. In this manner, this island model would be different then performing several random restarts of a single population model. The model would encourage higher rates of survivability that would make the search more efficient, possibly allowing population sizes to be reduced. Furthermore, this model would promote a broader exploration of the search space.

7.5.2 Similar Models

A similar model was proposed by Bessaou et al. (2000) for multimodal optimisation with genetic algorithms. In their model, a multipopulation algorithm performed migration between subpopulations. The subpopulations were then merged and individuals were redistributed to new subpopulations based on a speciation tree method that places genetically similar individuals together. A local genetic algorithm was then run on each island for a number of generations to serve as an 'intensification' phase. Our proposed model does not consider migration or the concept of merging the subpopulations before redistributing them according to species. The genetic algorithm species conserving model (Li et al., 2002) identified diverse individuals and assigned the fitter to be seeds which are propagated to the next generation. However, recombining diverse individuals, as seen in Section 7.3, may not always be ideal for genetic programming.

Fitness sharing over structures was used to balance the search between structure and contents (Hu et al., 2002). However, like other fitness sharing methods in genetic programming based on genetic dissimilarity, this approach ignores the possible benefits of homology and the possible negative effects of recombination using diverse parents. Lastly, the method by Potter and DeJong (2000) for coevolution of components requires each species to contribute toward fitness to survive and new

species are added when no improvements are found. In a similar way, future research with the proposed model can investigate different ways to remove stagnate islands while still allowing the possibility of variation to arise during stasis.

7.5.3 The Island, Post-Speciation Event and Other Comments

This investigation and proposal did not address the speciation event and subsequent search on the new species for two reasons. Firstly, the method to initialise a new subpopulation based on the species is problem dependent in the size of the search space around that species. For example, depending on the size of the individuals, the size of the function and terminal sets and the current distribution of individuals, different methods would be appropriate. Given a large function and terminal set, considering a species as sharing the exact same tree structure may be infeasible, let alone similar tree structures. In this case, a species may define an exact tree structure plus some commonality amongst the node labels. Adapting the subpopulation size is also a possible consideration in this case.

Secondly, the method of initialisation is dependent upon the operator currently being used. Given the use of an operator like standard subtree crossover, it makes little sense to expect the search to take place near the root of the trees when this operator is biased toward the leaves. Thus, initialising the new species based on the seed might focus the changes near the leaves first, then toward the root after a sufficient level of diversity is found. More homologous operators may be able to use a looser definition of species due to a lack of bias toward a particular area of the tree. In some cases, it may be good to just perform a phase of local search on the diverse individuals which would normally define a new species. In fact, this is a good way to initially validate this model. The local optimisation technique would probably need to be defined for each problem instance. According to the results presented in this thesis, the canonical genetic programming system routinely produces outlier individuals that would seem to be ideal candidates for further examination. While previous methods may have leveraged these individuals in more indirect ways, the proposed model intends to leverage these individuals directly.

What is not addressed in this study is the composition of the in-lier space. While previous research has shown the strong convergence characteristic of the population with respect to structure and content, the in-lier space may indeed be composed of several distinct clusters of genetically similar individuals. Figure 7.11 demonstrates how the in-liers could be composed of four distinct genetically similar clusters, instead of the single cluster shown in Figure 7.1. However, the actual identification of these spaces and distinct clusters is a complex issue requiring specific measures and methods.

The proposed model was largely motivated by the momentum in the literature toward the impor-

tance of structure, e.g. (Daida et al., 2003b), and diversity (as seen in earlier chapters). However, while the model proposed is general (i.e. explicitly considering outliers should generally improve results), the exact performance and tuning of model parameters will be problem specific.

7.6 Summary

The research presented in this chapter examined the role diversity has in guiding the search. An empirical analysis examined the role of the most dissimilar and fit individuals in the population. This was accomplished by studying the survivability of these individuals. Also, due to the many distributed models and methods proposed to improve search by means of increased diversity, this research also considered the role of the migrant.

One might expect that higher amounts of diversity generally leads to better search in genetic programming. Diversity and dissimilar individuals could provide key variations that allow new solutions to be discovered and improved by passing on their genetic material to offspring. While the results presented in this chapter showed how dissimilar individuals play an important role in producing offspring when the population is genetically similar, this role is usually unstable. Dissimilar individuals were particularly important when the many fit individuals also had the same fitness. That is, when the most fit subpopulation contains low phenotype diversity, dissimilar individuals are more successful in producing offspring that later contribute toward search. Also, as phenotype diversity is lost among the most fit individuals, the worse fit individuals are more successful in contributing new solutions to later generations.

These results raise questions about the motivation of methods based on dissimilarity mating and blindly promoting high diversity. If dissimilar individuals have a low chance of success producing offspring with high relative fitness, why concentrate effort on producing additional diversity? Lastly, the results also highlighted the possible ineffectiveness of migrants in distributed models to contribute positively toward search. While this was not directly investigated, the results certainly suggest further work should be carried out in validating the effectiveness of migrants.

A niche for island models was proposed that leverages the results from this chapter. The niche consists of identifying structurally different individuals (that are unlikely to mate effectively with the rest of the population) and speciates them to a new island. The new island provides a chance to search more effectively with this individual and the region of the search space this individual represents.

The conclusions of this thesis are presented in the following chapter.

CHAPTER 8

CONCLUSIONS

Genetic programming is a search heuristic that represents candidate solutions as computer programs. As such, genetic programming is used as a technique for optimisation, automatic programming and model induction. Like other metaheuristic methods, it combines a search strategy with operators and an objective function. However, genetic programming also uses a complex, variable-length representation, a population of solutions and a search strategy based on natural evolution. These elements combine to make analysis difficult.

When genetic programming does not produce desirable results, premature convergence and loss of diversity within the population are often blamed. Without diversity the algorithm cannot produce variation and move out of local optima due to convergence. However, the population rarely loses all diversity. Therefore, it is often misleading to attribute run failure to convergence. In fact, genetic programming runs are usually pre-determined to converge due to the representation and operators. Run failure is more likely due to the type and rate of convergence, not convergence itself.

The research presented in this thesis has contributed toward an improved understanding of genetic programming by focusing on the issue of diversity. While diversity loss can describe convergence and the inability to escape local optima, diversity also influences most decisions in the evolutionary process. After all, evolutionary algorithms are a population-based search method that rely on the population to provide search direction.

The results of this research demonstrate many ways to improve search efficiency and effectiveness in genetic programming. Improving this population-based search method has wide reaching benefits, demonstrated by the many applications of genetic programming, from artificial intelligence (mobile robotics, automatic programming and machine learning) to operations research and applied optimisation.

8.1 Contributions

This thesis has made the following contributions.

8.1.1 A survey and analysis of diversity in genetic programming demonstrated the complexity behind the issue of diversity measures and methods and the relationship between diversity and fitness.

Several measures of diversity frequently used in the literature were surveyed and analysed in Chapter 4. Specifically, the most typical measures from the literature referring to genotype and phenotype traits were used in an experimental study. The general behaviour of these measures showed the initial difficulty in assessing and attributing run failure to diversity loss as many measures behaved unexpectedly. For example, the measure of unique genotypes typically remained at high levels after only a few generations and never increased or decreased significantly. As individuals became larger in size, maintaining distinct individuals was not generally difficult.

In Chapter 4, using a measure based on edit distance, populations were seen to generally lose most diversity early in the run and then remain at low levels. There was not a distinct phase transition of diversity loss that could be attributed to an expected time when the run would become stuck in local optima. Rather, low edit distance diversity is the result of selection, the representation and operator. In some cases, the increased loss of diversity was linked to improved search performance, demonstrated by the correlation between good fitness and low diversity.

Measures of particular importance were those based on fitness values. A high number of unique phenotypes and high entropy were both correlated with the best fitness found during the evolutionary process. Problems with discrete fitness spaces in particular, such as the Ant and Parity problems, cause selection methods to become more random when the population loses unique fitness values. The loss of unique fitness values can also increase the chance that selection will pick deceptive or conflicting individuals to search with. This explains why high fitness-based diversity measures correlated well with good fitness.

The previous methods used to control diversity, like those used to control code growth, are often heavy-handed. Such methods are likely to effect the run in many unexpected ways. Therefore, in Chapter 5, a measure of diversity that incorporated ancestry information and inheritance was used.

8.1.2 An analysis using genetic lineages showed how a search metaphor of hill-climbing can be used to explain and improve genetic programming search. Also, the sampling of unique structures and behaviours demonstrated the the low sampling of both complex behaviours and unique structures of large size.

Genetic lineages were used to bias the selection method toward fit individuals from randomly chosen lineages. The results showed that genetic diversity was increased while producing smaller solutions, but with worse fitness on two of the three problem domains. A literature review of similar and related problem domains revealed that the results of using lineage selection clearly demonstrated the similarities between genetic programming and hill-climbing. When genetic programming benefits from a hill-climbing type search, increasing diversity worsened fitness by preventing a hill-climbing search from being carried out effectively. When genetic programming was likely to suffer from deception, the increased diversity of lineage selection improved fitness.

The idea that genetic programming may carry out a similar search to that of hill-climbing motivated an analysis of the sampling of structures and behaviours. In this case, a structure refers to a tree without any node contents. A behaviour was defined for each problem that reflected the fitness or complexity of solutions. These results demonstrated that genetic programming samples fewer unique structures of large sizes. Instead, when large structures were produced, the search effort was spent sampling different structures of an intermediate size. Problem specific results highlighted many areas where methods could be used to improve search performance.

The issue of code growth and bloat has received much attention in the genetic programming community. The mechanics of subtree crossover and the representation are thought to be responsible for code growth in genetic programming. However, there has been little effort to explain the varied rates of growth that occur in most experiments.

8.1.3 A causal model was developed which linked increased rates of code growth to non-decreased selection pressure and to increased similarity within the population. Decreased selection pressure occurs when fitness-based diversity is lost, and increased similarity in the population is the result of both faster convergence and non-decreased selection pressure.

To address the topic of varied rates of code growth, Chapter 6 investigated the change of population dynamics occurring under increasingly difficult problem instances. Using previously researched problem instances and several randomly generated instances, Chapter 6 showed how problem difficulty induces different kinds of population dynamics. These dynamics affect both selection pressure and diversity, and subsequently *the rate* of code growth. A causal model was also supported by a constructed problem and related literature. Results suggested that controlling the rate of code

growth can be achieved by considering the variability of selection pressure and the presence of diversity. Whereas previous methods are often heavy-handed in penalising the large individuals, this chapter outlined alternative ways in which code growth might be controlled without destructively effecting fitness improvements. Also, this chapter further highlighted the different types of behaviours of increasingly difficult problem instances.

Given the diversity results of Chapter 4, the results of controlling diversity in Chapter 5 and the effects of diversity on code growth and selection in Chapter 6, the next line of research carried out was to analyse the contribution of dissimilar individuals during search. Distributed models are frequently suggested as the remedy for convergence. As these models typically move migrants between subpopulations, this study allowed the role of these migrants to be examined.

8.1.4 An analysis using the Tree-String problem showed the inability to produce good offspring by both dissimilar-and-fit individuals and by similar-and-equally-well-fit individuals.

The definition of genetic outlier individuals, which are genetically dissimilar and highly fit, demonstrated that dissimilar and fit individuals had sporadic and variable survivability. This study showed that genetic programming populations consistently contain highly fit and dissimilar individuals that are often not considered in search due to their small numbers and their inability to produce fit offspring. The analyses of the in-lier population showed that in-liers contribute more consistently to future populations until they become genetically similar with many equal fitness values. When in-liers had less dissimilarity, the un-fit population and the genetic outliers contributed more to search. Lastly, the Tree-String problem was introduced. This problem is thought to be tunably difficult with many features related to other domains.

8.1.5 A model was proposed that identifies dissimilar individuals and moves them to new islands where they can contribute to search more effectively.

A survey of distributed models showed that they rarely assign an explicit role to dissimilar individuals, which are frequently exchanged between subpopulations. The analysis of genetic outliers suggested that these individuals are likely to have low survivability. A model was proposed to detect genetic outliers and give them a more suitable environment in which to evolve. In this way, genetic outliers are considered as *species*. Based on the results presented here and in previous literature, this type of model would appear to have a high chance of success.

8.1.6 Summary

The research presented in this thesis provides an analysis of several key issues in genetic programming. First, selection pressure and subtree crossover success depend on the type of fitness-based and genetic diversity in the population. Secondly, the rates of code growth could be controlled by adaptively controlling both fitness-based and genetic diversity. Thirdly, the type of search genetic programming carries out can be described using a metaphor of hill-climbing and some knowledge of the problem domain. Finally, the inability of dissimilar and fit individuals, and similar and equally well fit individuals, to produce good offspring was shown.

8.2 Remarks and Problem Specific Conclusions

In Chapter 3, questions were proposed for this thesis to address. In response to those questions, the contributions of this thesis are restated in those terms. Following this, problem specific results that appear throughout the thesis are highlighted together.

Question 1: How can diversity be measured and controlled, and are there ideal levels of diversity?

Chapter 4 surveyed several measures and methods of diversity. An experimental study demonstrated the behaviour of several of these measures. The search process in genetic programming must find suitable structures on which to represent content. Thus, measures of diversity which provide an accurate depiction of the structure and content in a population, such as edit distance diversity, are likely to capture important dynamics. However, a main driving force behind the algorithm is selection and recombination. Therefore, diversity measures which capture the property of fitness distributions will indicate the population's effect on selection pressure.

Many methods have been used to control diversity. Chapter 5 demonstrated the use of lineage selection and its good and bad effects on performance. The concept of genetic lineages can be an accurate indication of diversity loss in a canonical genetic programming system. Some methods encourage high genetic diversity or fitness-based diversity, while others are adaptive. However, as seen in Chapter 6, changing population diversity will also change other dynamics in the algorithm. The lack of a clear correlation between diversity and fitness in Chapter 4 emphasises the danger in assuming that high diversity (of a some type) leads to better fitness.

Question 2: Genetic programming is a population search method, thus, what effect does population diversity have on other aspects of the search process?

The effects of population diversity were seen in Chapter 6 when studying code growth. The results showed how low fitness diversity reduces selection pressure and how high genetic diversity can reduce code growth. Chapter 7 showed how search using populations with a high number of equally fit and similar individuals is guided more by the dissimilar and poorly fit individuals. Chapter 5 demonstrated the increased exploration ability of more genetically dissimilar populations, but at the cost of fitness. Chapter 5 suggested population convergence can promote a hill-climbing behaviour that is beneficial to improving fitness on some problems.

Question 3: What specific role does diversity play in the evolutionary process, i.e. do dissimilar individuals contribute offspring differently than the rest of the population?

In a general sense, dissimilar individuals allow for exploration to take place. However, as seen in Chapter 7, dissimilar individuals are not consistently effective in populations where they are likely to breed with very different individuals. However, as the population becomes more equivalent in fitness, dissimilar individuals provide a chance for escaping local optima. The previous literature suggests that diverse individuals would be most effective when breeding in separate populations, where they compete and recombine with more *like* individuals. One of the most important areas of improvement that can be made to genetic programming, or any population-based search method, is the precise characterisation of the population components to allow for new methods to improve dynamics, efficiency and the rate of success. The research presented here makes a contribution to that effort.

The majority of experiments used in this thesis involved three problem domains commonly used in the genetic programming literature. The reason for this was twofold. Firstly, it was necessary to use the same problem domains, and algorithm parameters, on which to build sound explanations across the different chapters and experiments. Secondly, as these problems are frequently used by the community to develop new methods and theoretical arguments, it is useful to apply to them the full range of the diversity experiments and analysis for future reference.

8.2.1 Ant Remarks

The results from Chapter 5 showed an additional example of how avoiding deception, via increased diversity, improved performance. The results of the sampling study at the end of Chapter 5 also demonstrated how the Ant problem samples many different structures of a small size and fewer different behaviours of increasing fitness.

The correlation between good fitness and high fitness-based diversity in the Chapter 4 Ant experiments also showed the benefit of maintaining more different fitness values, i.e. fewer different behaviours with the same fitness. However, the correlation between good fitness and low edit distance diversity in the later stages of the Ant experiments showed how better runs also converged more. The survivability study at the end of Chapter 7 showed a phase change in the Ant experiments, where in-liers began producing the majority of surviving offspring. This change, around generation 10-15, occurred when more and more individuals became part of the un-fit population. When better runs converge, and in-liers have much higher survivability, it is not clear what the rest of the population contributes. A hill-climbing or local optimisation strategy at this point may be a more effective and efficient way to continue search.

8.2.2 Parity Remarks

In Chapter 5, lineage selection prevented a hill-climbing-like behaviour that increased genetic diversity but led to worse fitness. Previous research showed how elitist strategies that incorporated an element of diversity control also successfully improved search. The deception in the Parity problem is due to the many different solutions that have the same fitness. This is in contrast to the Ant problem, where deception was also due to the fact that some solutions are not easily improved by genetic programming. The sampling study in Chapter 5 showed the ease of which genetic programming can sample solutions near the random strategy, and the difficulty in sampling better and worse ones. The benefit of elitism would focus the search more consistently on a particular strategy and create a hill-climbing behaviour of search.

Chapter 7 showed a phase change near generation 20 in the Parity experiments, where further populations came mostly from the in-lier population. This is similar to the Ant experiments. The variable and sporadic survival rates in the genetic outlier population suggests the importance of variation when the population loses diversity, but also shows the inability of the genetic outliers to produce viable offspring consistently. The ability to distinguish between the many near random behaviours would help to reduce the effects of deception in this problem and allow a more predictable selection pressure.

8.2.3 Regression Remarks

The range of symbolic regression instances used throughout this thesis provided a set of very different behaviours. From the very difficult, the Rastrigin function, to much easier ones, the Quartic and Binomial-3 functions, diversity measures did not capture a similar dynamic in previous problems that was important to search. One reason for this is that the continuous fitness space in Regression

experiments allows genetic programming to easily maintain high fitness-based diversity. This can most easily be seen in the phenotype and entropy measures in Chapter 4 that remained at much higher levels than other domains.

The Tree-String problem from Chapter 7 would appear to be most closely related to Regression problems with its dual and often conflicting objectives of structure and content. Applying the results from the Tree-String experiments to the Regression problem suggests that much of the future populations come from diverse individuals that may not necessarily be the most fit in the population. That is, in the Tree-String experiments, when considering the definition of outliers as better-than or equivalent-to more than half the population, the outliers contributed a considerably higher rates of survivability than selection. The results of the Binomial-3 instance in Chapter 7 also show that outliers play an important part in the search process.

8.3 Future Directions

The empirical investigations in this thesis suggest many possible directions for future research.

8.3.1 Diversity Measures and Methods

Developing measures based on the operators would give an improved understanding of diversity and of the fitness landscape. However, the complexity of the representation would make these measures computationally expensive. Approximate measures based on the operators would require careful consideration of their accuracy. Intermediate data structures could be used improve the efficiency of such complex measures.

Chapter 5 used a new diversity method based on genetic lineages. Additional research with this method could yield more efficient ways to control diversity and search, or easily simulate mating, niching and island models. Chapter 6 suggested controlling diversity by adapting selection pressure, which could lead to slower and more controlled code growth without fewer negative effects to fitness. Future work could also characterise the "effective" selection pressure that populations with varying diversity induce.

8.3.2 Code Growth and Problem Difficulty

The results in Chapter 6 demonstrated several possible areas of controlling code growth indirectly. While the recombination of genetically similar individuals is likely to cause a consistent amount of code growth in future populations, the recombination of dissimilar individuals, while producing

less overall code growth, is more likely to also produce fewer good offspring. Therefore, investigating adaptive recombination methods that are aware of the similarity of individuals could be a solution to minimising unjustified code growth while achieving good fitness.

Chapter 6 also showed the effects of problem difficulty. It may be possible to control population diversity to better deal with harder instances. For example, when a difficult instance causes a population to contain one really good individual, methods which dynamically prevent the over-selection of this individual are likely to improve the overall performance of the algorithm.

Parts of the causal model in Chapter 6 remain to be fully evaluated. An area of future research here would be measuring the dissimilarity of solutions in hard and easy instances. The hypothesis stated that easy instances allow more optimal and different solutions to be acquired quickly. This was based on the notion that easy instances can be solved equally well by different solutions in different ways. Hard instances were thought to be solved by fewer solutions that are more similar. Future work can investigate the actual dissimilarities in solutions for easy and hard instances.

8.3.3 Defining the Role of the Populations

This thesis has provided a better understanding of the role of the genetic programming population. However, several areas of the research in Chapter 7 could be further examined. For example, the exact dissimilarity between mating parents, the structure of dissimilarity in the un-fit population and in-depth analysis of survivability and genetic outliers in other domains are potential areas of future research.

If genetic programming is considered as a stochastic hill-climber using a procedural representation, the role of the population is to provide genetic material in a more elitist algorithm. If genetic programming is to use the population to sufficiently explore the landscape in parallel, one needs to explicitly maintain portions of the population in beneficial areas and dispense sufficient search effort appropriately. The proposed niche for island models is an obvious extension to the research presented here and gives the population a much clearer role during search. Empirical studies, some of which were described in detail in Chapter 7, need to be carried out to validate the proposed island model.

Chapter 7 also introduced the Tree-String problem. Investigating the tunable nature of this problem would validate it as a future testbed for understanding complex domains. Future work can begin by modifying the size of the string symbol set and the method to grow target tree structures.

8.3.4 Extended Metaphors of Search

A metaphor of hill-climbing was used in Chapter 5 to characterise genetic programming search. Developing accurate models and metaphors of genetic programming will allow more intuitive improvement and analysis of results. The metaphor used here helped to understand these results as well as previous results. Further development and research along these lines may yield surprising and extremely useful models for the community. Specifically, one could build intermediate models between stochastic hill-climbers (O'Reilly and Oppacher, 1994; Juels and Wattenberg, 1995) and canonical genetic programming. These intermediate models would contain various properties of both methods and should help show more clearly why and how the two methods carry out search differently.

Bibliography

Adamidis, P. and Petridis, V. (1996). Co-operating populations with different evolution behavior. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 188–191, Nagoya, Japan.

Altenberg, L. (1994). Emergent phenomena in genetic programming. In Sebald, A. and Fogel, L., editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 233–241. World Scientific.

Andre, D. and Koza, J. (1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In Angeline, P. and Kinnear, Jr., K., editors, *Advances in Genetic Programing 2*, chapter 16. The MIT Press, Cambridge, MA, USA.

Angeline, P. (1997). Subtree crossover: Building block engine or macromutation? In Koza, J. et al., editors, *Proceedings of the Second Annual Genetic Programming Conference*, pages 9–17, Stanford University, USA. Morgan Kaufmann.

Angeline, P. J. (1998). A historical perspective on the evolution of executable structures. *Informaticae*, 36(1-4):179–195.

Bäck, T., Fogel, D., and Michalewicz, Z., editors (2000a). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, UK.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (2000b). *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, Bristol, UK.

Banzhaf, W. and Langdon, W. B. (2002). Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming: An Introduction*. Morgan Kaufmann, Inc., San Francisco, USA.

Barr, A., Cohen, P., and Feigenbaum, E., editors (1989). *The Handbook of Artificial Intelligence*, volume 4. Addison-Wesley, Reading, MA.

Belding, T. (1995). The distributed genetic algorithm revisited. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 114–121, San Francisco, CA. Morgan Kaufmann.

Bersano-Begey, T. (1997). Controlling exploration, diversity and escaping local optima in GP. In Koza, J., editor, *Late Breaking Papers at the Genetic Programming Conference*, pages 7–10, Stanford University, CA.

Bessaou, M., Pétrowski, A., and Siarry, P. (2000). Island model cooperating with speciation for multimodal optimization. In Schoenauer, M. et al., editors, *Parallel Problem Solving from Nature*, pages 437–446, Paris, France. Springer Verlag.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland.

Bongard, J. C. (1999). Coevolutionary dynamics of a multi-population genetic programming system. In Floreano, D., Nicoud, J.-D., and Mondada, F., editors, *Proceedings of the 5th European Conference on Advances in Artificial Life*, volume 1674 of *LNAI*, pages 154–158, Berlin. Springer.

Brameier, M. and Banzhaf, W. (2002). Explicit control of diversity and effective variation distance in linear genetic programming. In Tettamanzi, A. et al., editors, *Genetic Programming, Proceedings of the 5th European Conference*, volume 2278 of *LNCS*, pages 162–171, Kinsale, Ireland. Springer-Verlag.

Bremermann, H. (1962). Optimization through evolution and recombination. In Yovits, M., Jacobi, G. T., and Goldstine, G., editors, *Self-Organizing Systems*, pages 93–106. Spartan Books, Washington DC.

Burke, D., Jong, K. D., Grefenstette, J., Ramsey, C., and Wu, A. (1998). Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6(4):387–410.

Burke, E., Gustafson, S., and Kendall, G. (2002a). A survey and analysis of diversity measures in genetic programming. In Langdon, W. B. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716–723, New York. Morgan Kaufmann Publishers.

Burke, E., Gustafson, S., and Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62.

Burke, E., Gustafson, S., Kendall, G., and Krasnogor, N. (2002b). Advanced population diversity measures in genetic programming. In Guervós, J. M. et al., editors, *Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 341–350, Granada, Spain. Springer.

Burke, E., Gustafson, S., Kendall, G., and Krasnogor, N. (2003). Is increasing diversity in genetic programming beneficial? An analysis of the effects on fitness. In McKay, B. et al., editors, *Congress on Evolutionary Computation*, pages 1398–1405, Canberra, Australia. IEEE Press.

Cantú-Paz, E. (1999). Topologies, migration rates, and multi-population parallel genetic algorithms. In Banzhaf, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 91–98, San Francisco, CA. Morgan Kaufmann.

Cohoon, J., Hegde, S., Martin, W., and Richards, D. (1987). Punctuated equilibria: a parallel genetic algorithm. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, Hillsdale, NJ, USA. Lawrence Erlbaum Associates.

Collins, R. (1992). *Studies in Artificial Evolution*. Ph.D. dissertation, Department of Computer Science, University of California at Los Angeles.

Cramer, N. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J., editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA.

Daida, J. (2002). Limits to expression in genetic programming: Lattice-aggregate modeling. In Fogel, D. et al., editors, *Congress on Evolutionary Computation*, pages 273–278, Honolulu, USA. IEEE Press.

Daida, J., Bertram, R., Stanhope, S., Khoo, J., Chaudhary, S., Chaudhri, O., and Polito II, J. (2001). What makes a problem GP-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165–191.

Daida, J., Hilss, A., Ward, D., and Long, S. (2003a). Visualizing tree structures in genetic programming. In Cantú-Paz, E. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2724 of *LNCS*, pages 1652–1664, Chicago, IL, USA. Springer-Verlag.

Daida, J., Li, H., Tang, R., and Hilss, A. (2003b). What makes a problem GP-hard? validating a hypothesis of structural causes. In Cantú-Paz, E. et al., editors, *Proceedings of the Genetic and Evolutionary Computation*, volume 2724 of *LNCS*, pages 1665–1677, Chicago, IL, USA. Springer-Verlag.

Darwen, P. and Yao, X. (2000). Does extra genetic diversity maintain escalation in a co-evolutionary arms race. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):191–200.

Darwen, P. and Yao, X. (2001). Why more choices cause less cooperation in iterated prisoner's dilemma. In *Proceedings of the Congress on Evolutionary Computation*, pages 987–994, Seoul, Korea. IEEE Press.

Darwin, C. (1859). *The Origin of Species by Means of Natural Selection*. Mentor Reprint, 1958, NY.

Davis, L., editor (1991). *Handbook of Genetic Algorithms*. International Thomson Press, Boston, MA.

de Jong, E., Watson, R., and Pollack, J. (2001). Reducing bloat and promoting diversity using multi-objective methods. In Spector, L. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, San Francisco, CA. Morgan Kaufmann.

Deb, K. and Goldberg, D. (1989). An investigation of niche and species formation in genetic function optimization. In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, CA, USA. Morgan Kaufmann.

DeJong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. thesis, Department of Compter and Communication Sciences, University of Michigan.

D'haeseleer, P. (1994). Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, FL, USA. IEEE Press.

D'haeseleer, P. and Bluming, J. (1994). Effects of locality in individual and population evolution. In Kinnear, Jr., K., editor, *Advances in Genetic Programming*, chapter 8, pages 177–198. MIT Press.

Edmonds, B. (2001). Learning appropriate contexts. In Akman, V. et al., editors, *Modelling and Using Context*, volume 2116 of *LNAI*, pages 143–155. Springer-Verlag.

Eggermont, J. and van Hemert, J. (2001). Adaptive genetic programming applied to new and existing simple regression problems. In Miller, J. et al., editors, *Genetic Programming, Proceedings of the 4th European Conference*, volume 2038 of *LNCS*, pages 23–35, Lake Como, Italy. Springer-Verlag.

Eiben, A. E. and Schippers, C. A. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50.

Eiben, G. and van Hemert, J. (1999). SAW-ing EAs: Adapting the fitness function for solving constrained problems. In Corne, D. et al., editors, *New Ideas in Optimization*, pages 389–402. McGraw-Hill, London.

Ekárt, A. (2000). Shorter fitness preserving genetic programs. In Fonlupt, C. et al., editors, *Artificial Evolution. 4th European Conference, Selected Papers*, volume 1829 of *LNCS*, pages 73–83, Dunkerque, France.

Ekárt, A. and Németh, S. (2000). A metric for genetic programs and fitness sharing. In Poli, R. et al., editors, *Genetic Programming, Proceedings of the 3rd European Conference*, volume 1802 of *LNCS*, pages 259–270, Edinburgh. Springer-Verlag.

Ekárt, A. and Németh, S. (2001). Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73.

Ekárt, A. and Németh, S. (2002). Maintaining the diversity of genetic programs. In Foster, J. et al., editors, *Genetic Programming, Proceedings of the 5th European Conference*, volume 2278 of *LNCS*, pages 162–171, Kinsale, Ireland. Springer-Verlag.

Eldredge, N. and Gould, S. (1972). *Punctuated Equilibria: An Alternative to Phyletic Gradualism*, chapter 5, pages 82–115. Freeman, Cooper and Co.

Eshelman, L. and Schaffer, J. (1993). Crossover's niche. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 9–14, San Mateo, CA. Morgan Kaufman.

Fernandes, C. and Rosa, A. (2001). A study on non-random mating and varying population size in genetic algorithms using a royal road function. In *Proceedings of the Congress on Evolutionary Computation*, pages 60–66. IEEE Press.

Fernandez, F., Tomassini, M., Punch, W., and Sanchez, J. M. (2000). Experimental study of isolated multipopulation genetic programming. In Whitley, D. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, page 536, Las Vegas, NV, USA. Morgan Kaufmann.

Fernandez, F., Tomassini, M., and Vanneschi, L. (2001). Studying the influence of communication topology and migration on distributed genetic programming. In Miller, J. et al., editors, *Genetic Programming, Proceedings of the 4th European Conference*, volume 2038 of *LNCS*, pages 51–63, Lake Como, Italy. Springer-Verlag.

Fernandez, F., Tomassini, M., and Vanneschi, L. (2003). An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51.

Fogel, D. (1998). *Evolutionary Computation: The Fossil Record*. IEEE Press, Piscataway, NJ.

Fogel, L., Owens, A., and Walsh, M. (1966). *Aritifial Intelligence Through Simulated Evolution*. John Wiley & Sons, Inc., New York.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. *Aust. J. of Biol. Sci.*, 10:484–491.

Friedberg, R. (1958). A learning machine: Part i. *IBM Journal of Research and Development*, (2):2–13.

Friedberg, R., Dunham, B., and North, J. (1959). A learning machine: Part ii. *IBM Journal of Research and Development*, (3):282–287.

Gathercole, C. and Ross, P. (1996). An adverse interaction between crossover and restricted tree depth in genetic programming. In Koza, J. et al., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, CA, USA. MIT Press.

Geard, N. and Wiles, J. (2002). Diversity maintenance on neutral landscapes: An argument for recombination. In Fogel, D. et al., editors, *Proceedings of the Congress on Evolutionary Computation*, pages 211–213, Honolulu, USA. IEEE Press.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549.

Glover, F. and Kochenberger, G., editors (2003). *Handbook of Metaheuristics*. Kluwer, Boston, MA.

Glover, F. and Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers, Boston, USA.

Goldberg, D. (2002). *The Design of Innovation, Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Boston, MA.

Goldberg, D. and O'Reilly, U.-M. (1998). Where does the good stuff go, and why? how contextual semantics influence program structure in simple genetic programming. In Banzhaf, W. et al., editors, *Genetic Programming, Proceedings of the First European Workshop*, volume 1391 of *LNCS*, pages 16–36, Paris. Springer-Verlag.

Goldberg, D. and Richardson, J. (1987). Genetic algorithms with sharing for multimodalfunction optimization. In Grefenstette, J., editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, pages 41–49, Cambridge, MA. Lawrence Erlbaum Associates.

Gustafson, S., Burke, E., and Kendall, G. (2004a). Sampling of unique structures and behaviours in genetic programming. In Keijzer, M. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, Coimbra, Portugal. Springer-Verlag.

Gustafson, S., Ekárt, A., Burke, E., and Kendall, G. (expected to appear 2004b). Problem difficulty and code growth in genetic programming. *Genetic Programming and Evolvable Hardware*.

Gustafson, S. and Krasnogor, N. (2003). Visualising populations of rooted labeled trees on a lattice. Unpublished. http://www.cs.nott.ac.uk/~smg/.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Holland, J. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation*, 8(4):373–391.

Hu, J. and Goodman, E. (2002). The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In Fogel, D. et al., editors, *Proceedings of the Congress on Evolutionary Computation*, pages 49–54, Honolulu, USA. IEEE Press.

Hu, J., Seo, K., Li, S., Fan, Z., Rosenberg, R., and Goodman, E. (2002). Structure fitness sharing (SFS) for evolutionary design by genetic programming. In Langdon, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 780–787, New York. Morgan Kaufmann Publishers.

Hutter, M. (2002). Fitness uniform selection to preserve genetic diversity. In Fogel, D. et al., editors, *Proceedings of the Congress on Evolutionary Computation*, pages 783–788, Honolulu, USA. IEEE Press.

Iba, H. (1996). Random tree generation for genetic programming. In Voigt, H.-M. et al., editors, *Parallel Problem Solving from Nature*, volume 1141 of *LNCS*, pages 144–153, Berlin, Germany. Springer Verlag.

Iba, H., de Garis, H., and Sato, T. (1994). Genetic programming using a minimum description length principle. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 12, pages 265–284. MIT Press.

Igel, C. and Chellapilla, K. (1999). Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In Banzhaf, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1061–1068, Orlando, FL, USA. Morgan Kaufmann.

Iwashita, M. and Iba, H. (2002). Island model GP with immigrants aging and depth-dependent crossover. In Fogel, D. et al., editors, *Proceedings of the Congress on Evolutionary Computation*, pages 267–272, Honolulu, USA. IEEE Press.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Korf, M. F. R., Taylor, C., and Wang, A. (1991). Evolution as a theme in artificial life: The genesys/tracker system. In Langton, C. et al., editors, *Proceedings of Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity*, volume X. Addison-Wesley.

Juels, A. and Wattenberg, M. (1995). Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical Report Technical Report CSD-94-834. Computers Science Department, University of California at Berkeley, USA.

Juillé, H. and Pollack, J. (1996). Massively parallel genetic programming. In Angeline, P. and Kinnear, Jr., K., editors, *Advances in Genetic Programming 2*, pages 339–358. The MIT Press, Cambridge, MA, USA.

Kauffman, S. (1993). *The origins of order: self-orginiation and selection in evolution*. Oxford University Press, New York, NY.

Keijzer, M. (1996). Efficiently representing populations in genetic programming. In Angeline, P. and Kinnear, Jr., K., editors, *Advances in Genetic Programming 2*, chapter 13, pages 259–278. MIT Press, Cambridge, MA, USA.

Keijzer, M. (2002). *Scientific Discovery using Genetic Programming*. PhD thesis, Danish Technical University, Lyngby, Denmark.

Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, C. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, volume 2610 of *LNCS*, pages 71–83, Essex, UK. Springer-Verlag.

Keller, R. and Banzhaf, W. (1995). Explicit maintenance of genetic diversity on genospaces. Internal Report, University of Dortmund.

Keller, R. and Banzhaf, W. (1996). Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In Koza, J. et al., editors, *Proceedings of First Annual Conference on Genetic Programming*, pages 116–122, Stanford University, CA, USA. MIT Press.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680.

Kishore, J., Patnaik, L., Mani, V., and Agrawal, V. (2001). Genetic programming based pattern classification with feature space partitioning. *Information Sciences*, 131(1-4):65–86.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, J. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge MA, USA.

Koza, J., Andre, D., Bennett III, F., and Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., and Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Krasnogor, N. (2002). *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England, Bristol, UK.

Langdon, W. (1998a). *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston.

Langdon, W. (1998b). The evolution of size in variable length representations. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, AL, USA. IEEE Press.

Langdon, W. (1999). Scaling of program fitness spaces. *Evolutionary Computation*, 7(4):399–428.

Langdon, W. (2000a). Quadratic bloat in genetic programming. In Whitley, D. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 451–458, Las Vegas, NV, USA. Morgan Kaufmann.

Langdon, W. (2000b). Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119.

Langdon, W. and Poli, R. (1998a). Fitness causes bloat: Mutation. In Banzhaf, W. et al., editors, *Genetic Programming, Proceedings of the 1st European Workshop*, volume 1391 of *LNCS*, pages 37–48, Paris. Springer-Verlag.

Langdon, W. and Poli, R. (1998b). Why ants are hard. In Koza, J. et al., editors, *Proceedings of the Third Annual Conference on Genetic Programming*, pages 193–201, Madison, WI, USA. Morgan Kaufmann.

Langdon, W. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer-Verlag, Berlin.

Langdon, W., Soule, T., Poli, R., and Foster, J. (1999). The evolution of size and shape. In Spector, L. et al., editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA.

Li, J.-P., Balazs, M., Parks, G., and Clarkson, P. (2002). A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234.

Lin, S.-C., Punch, W., and Goodman, E. (1994). Coarse-grain genetic algorithms, categorization and new approaches. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37, Dallas, TX, USA. IEEE Computer Society Press.

Liu, Y., Yao, X., and Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387.

Loveard, T. (2003). Genetic programming with meta-search: Searching for a successful population within the classification domain. In Ryan, C. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, volume 2610 of *LNCS*, pages 121–131, Essex, UK. Springer-Verlag.

Lucas, J., van Baronaigien, D., and Ruskey, F. (1993). On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366.

Luke, S. (1998). Genetic programming produced competitive soccer softbot teams for robocup97. In Koza, J. et al., editors, *Proceedings of the Third Annual Conference on Genetic Programming*, pages 214–222, Madison, WI, USA. Morgan Kaufmann.

Luke, S. (2000). Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283.

Luke, S. (2001). When short runs beat long runs. In Spector, L. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 74–80, San Francisco, CA, USA. Morgan Kaufmann.

Luke, S. (2003). Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106.

Luke, S. (2004). ECJ: A java-based evolutionary computation and genetic programming system. http://www.cs.umd.edu/projects/plus/ec/ecj/.

Luke, S. and Panait, L. (2001). A survey and comparison of tree generation algorithms. In Spector, L. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 81–88, San Francisco, USA. Morgan Kaufmann.

Luke, S. and Panait, L. (2002a). Fighting bloat with nonparametric parsimony pressure. In Guervós, J. M. et al., editors, *Parallel Problem Solving from Nature*, number 2439 in Lecture Notes in Computer Science, LNCS, pages 411–420, Granada, Spain. Springer-Verlag.

Luke, S. and Panait, L. (2002b). Lexicographic parsimony pressure. In Langdon, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, New York. Morgan Kaufmann Publishers.

Luke, S. and Spector, L. (1998). A revised comparison of crossover and mutation in genetic programming. In Koza, J. et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 208–213, San Francisco, CA. Morgan Kaufmann.

Martin, W. N., Lienig, J., and Cohoon, J. P. (2000). Island (migration) models: evolutionary algorithms based on punctuated equilibria. In Bäck, T. et al., editors, *Evolutionary Computation 2*, chapter 15. Institute of Physics Publishing, Bristol, UK.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

McKay, R. (2000). Fitness sharing in genetic programming. In Whitley, D. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 435–442, Las Vegas, NV, USA. Morgan Kaufmann.

McKay, R. and Abbass, H. (2001a). Anti-correlation: A diversity promoting mechanisms in ensemble learning. *The Australian Journal of Intelligent Information Processing Systems*, (3/4):139–149.

McKay, R. and Abbass, H. (2001b). Anticorrelation measures in genetic programming. In Kasabov, N. and Whigham, P., editors, *Australasia-Japan Workshop on Intelligent and Evolutionary Systems*, pages 45–51, Dunedin, New Zealand.

McPhee, N. and Hopper, N. (1999). Analysis of genetic diversity through population history. In Banzhaf, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1112–1120, FL, USA. Morgan Kaufmann.

Monsieurs, P. and Flerackers, E. (2003). Reducing population size while maintaining diversity. In Ryan, C. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, volume 2610 of *LNCS*, pages 145–156, Essex, UK. Springer-Verlag.

Montana, D. (1995). Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program report 826, California Institute of Technology, Pasadena, CA 91125, U.S.A.

Nienhuys-Cheng, S.-H. (1997). Distance between Herbrand interpretations: a measure for approximations to a target concept. In Lavraĉ, N. and Dẑeroski, S., editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of LNAI, pages 213–226, Prague, Czech Republic. Springer-Verlag.

Nikolaev, N. and Iba, H. (2001). Accelerated genetic programming of polynomials. *Genetic Programming and Evolvable Machines*, 2(3):231–257.

Nilsson, N. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York.

Nordin, P., Banzhaf, W., and Francone, F. D. (1999). Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In Spector, L. et al., editors, *Advances in Genetic Programming 3*, chapter 12, pages 275–299. MIT Press, Cambridge, MA, USA.

Olsson, R. (1995). Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74(1):55–81.

O'Reilly, U.-M. (1995). *An Analysis of Genetic Programming*. PhD thesis, Carelton University, Ottawa, Ontario, Canada.

O'Reilly, U.-M. (1997). Using a distance metric on genetic programs to understand genetic operators. In *IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*, volume 5, pages 4092–4097, FL, USA.

O'Reilly, U.-M. (1998). The impact of external dependency in genetic programming primitives. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 306–311, Anchorage, AL, USA. IEEE Press.

O'Reilly, U.-M. and Goldberg, D. (1998). How fitness structure affects subsolution acquisition in genetic programming. In Koza, J. et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 269–277, Madison, WI, USA. Morgan Kaufmann.

O'Reilly, U.-M. and Oppacher, F. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Davidor, Y. et al., editors, *Parallel Problem Solving from Nature*, number 866 in LNCS, pages 397–406, Jerusalem. Springer-Verlag.

O'Reilly, U.-M. and Oppacher, F. (1995). Hybridized crossover-based search techniques for program discovery. In *Proceedings of the World Conference on Evolutionary Computation*, volume 2, pages 573–578, Perth, Australia. IEEE Press.

O'Reilly, U.-M. and Oppacher, F. (1996). A comparative analysis of GP. In Angeline, P. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 2, pages 23–44. MIT Press, Cambridge, MA, USA.

Page, J., Poli, R., and Langdon, W. B. (1999). Smooth uniform crossover with smooth point mutation in genetic programming: A preliminary study. In Poli, R. et al., editors, *Genetic Programming, Proceedings of the 3rd European Conference*, volume 1598 of *LNCS*, pages 39–49, Goteborg, Sweden. Springer-Verlag.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial optimization : algorithms and complexity*. Prentice Hall, Englewood Cliffs, NJ.

Pardalos, P. and Resende, M., editors (2002). *Handbook of Applied Optimization*. Oxford University Press, New York, NY.

Pei, H. and Goodman, E. (2001). A comparison of cohort genetic algorithms with canonical serial and island-model distributed ga's. In Spector, L. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 501–510, San Francisco, CA, USA. Morgan Kaufmann.

Pettey, C., Leuze, M., and Grefenstette, J. (1987). A parallel genetic algorithm. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Hillsdale, NJ, USA. Lawrence Erlbaum Associates.

Pevzner, P. (2000). *Computational Molecular Biology An Algorithmic Approach*. MIT Press, Cambridge, MA, USA.

Platel, M., Clergue, M., and Collard, P. (2003). Maximum homologous crossover for linear genetic programming. In Ryan, C. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, volume 2610 of *LNCS*, pages 200–210, Essex, UK. Springer-Verlag.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In Ryan, C. et al., editors, *Genetic Programming, Proceedings of the 6th European Conference*, volume 2610 of *LNCS*, pages 200–210, Essex, UK. Springer-Verlag.

Poli, R. and Langdon, W. (1998a). On the search properties of different crossover operators in genetic programming. In Koza, J. et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 293–301, Madison, WI, USA. Morgan Kaufmann.

Poli, R. and Langdon, W. (1998b). Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252.

Poli, R. and McPhee, N. (2001). Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In Miller, J. et al., editors, *Genetic Programming, Proceedings of the 4th European Conference*, volume 2038 of *LNCS*, pages 126–142, Lake Como, Italy. Springer Verlag.

Poli, R. and McPhee, N. (2003a). General schema theory for genetic programming with subtree-swapping crossover: Part i. *Evolutionary Computation*, 11(1):53–66.

Poli, R. and McPhee, N. (2003b). General schema theory for genetic programming with subtree-swapping crossover: Part ii. *Evolutionary Computation*, 11(2):169–206.

Poli, R. and Page, J. (2000). Solving high-order boolean parity problems with smooth uniform crossover, sub-machine-code gp and demes. *Genetic Programming and Evolvable Machines*, 1:37–56.

Potter, M. and De Jong, K. (1994). A cooperative coevolutionary approach to function optimization. In Davidor, Y. et al., editors, *Parallel Problem Solving from Nature*, volume 866 of *LNCS*, pages 249–257, Berlin. Springer-Verlag.

Potter, M. and De Jong, K. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Provine, W. (1986). *Sewall Wright, Evolution, Selected Papers*. The University of Chicago Press, Chicago, IL, USA.

Punch, W. (1998). How effective are multiple populations in genetic programming. In Koza, J. et al., editors, *Proceedings of the Third Annual Conference on Genetic Programming*, pages 308–313, Madison, WI, USA. Morgan Kaufmann.

Punch, W., Zongker, D., and Goodman, E. (1996). The royal tree problem, a benchmark for single and multi-population genetic programming. In Angeline, P. and Kinnear, Jr., K., editors, *Advances in Genetic Programming 2*, chapter 15, pages 299–316. The MIT Press, Cambridge, MA, USA.

Rechenberg, I. (1965). *Cybernetic solution path of an experimental problem*. Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK.

Reeves, C., editor (1995). *Modern heuristic techniques for combinatorial problems*. McGraw-Hill, London.

Rosca, J. (1995a). Entropy-driven adaptive representation. In Rosca, J., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, Tahoe City, CA, USA.

Rosca, J. (1995b). Genetic programming exploratory power and the discovery of functions. In McDonnell, J. et al., editors, *Proceedings of the Fourth Conference on Evolutionary Programming*, pages 719–736, San Diego, CA. MIT Press.

Rosca, J. (1997a). Analysis of complexity drift in genetic programming. In Koza, J. et al., editors, *Proceedings of the Second Annual Genetic Programming Conference*, pages 286–294, Stanford University, CA. Morgan Kaufmann.

Rosca, J. (1997b). *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, Department of Computer Science, The College of Arts and Sciences, University of Rochester, Rochester, NY 14627, USA.

Rosca, J. and Ballard, D. (1995). Causality in genetic programming. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 256–263, Pittsburgh, PA, USA. Morgan Kaufmann.

Rosca, J. and Ballard, D. (1996). Discovery of subroutines in genetic programming. In Angeline, P. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 9, pages 177–202. MIT Press, Cambridge, MA, USA.

Rosca, J. and Ballard, D. (1999). Rooted-tree schemata in genetic programming. In Spector, L. et al., editors, *Advances in Genetic Programming 3*, chapter 11, pages 243–271. MIT Press, Cambridge, MA, USA.

Ryan, C. (1994). Pygmies and civil servants. In Kinnear, Jr., K., editor, *Advances in Genetic Programming*, chapter 11, pages 243–263. MIT Press, Cambridge, MA.

Ryan, C., Collins, J. J., and O Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In Banzhaf, W. et al., editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 83–95, Paris. Springer-Verlag.

Shapiro, S., editor (1990). *Encyclopedia of Artificial Intelligence*. Wiley, New York, NY.

Siegel, S. (1956). *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill Book Company, Inc., New York.

Smith, P. and Harries, K. (1998). Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation*, 6(4):339–360.

Smith, R. and Bonacina, C. (2003). Mating restriction and niching pressure: Results from agents and implications for general EC. In Cantú-Paz, E. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2724 of *LNCS*, pages 1382–1393, Chicago, IL, USA. Springer-Verlag.

Smith, R., Forrest, S., and Perelson, A. (1993). Searching for diverse, cooperative subpopulations with genetic algorithms. *Evolutionary Computation*, 1(2):127–149.

Soule, T. and Foster, J. (1997). Code size and depth flows in genetic programming. In Koza, J. et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA. Morgan Kaufmann.

Soule, T. and Foster, J. (1998). Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309.

Soule, T. and Heckendorn, R. (2002). An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(3):283–309.

Spector, L. (1996). Simultaneous evolution of programs and their control structures. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 7, pages 137–154. MIT Press, Cambridge, MA, USA.

Spector, L. and Luke, S. (1996). Cultural tansmission of information in genetic programming. In Koza, J. et al., editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 200–208, Stanford University, CA, USA. MIT Press.

Stoffel, K. and Spector, L. (1996). High-performance, parallel, stack-based genetic programming. In Koza, J. et al., editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 224–229, Stanford University, CA, USA. MIT Press.

Tackett, W. (1994). *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, USA.

Tackett, W. and Carmi, A. (1994). The donut problem: Scalability and generalization in genetic programming. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 7, pages 143–176. MIT Press, Cambridge, MA, USA.

Tanese, R. (1987). Parallel genetic algorithms for a hypercube. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 177–183, Hillsdale, NJ, USA. Lawrence Erlbaum Associates.

Tanese, R. (1989). Distributed genetic algorithms. In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, San Mateo, CA, USA. Morgan Kaufmann.

Tongchim, S. and Chongstitvatana, P. (1999). Speedup improvement on automatic robot programming by parallel genetic programming. In *Proceedings of IEEE International Symposium On Intelligent Signal Processing and Communication Systems*, pages 77–80, Thailand. IEEE Press.

Tongchim, S. and Chongstitvatana, P. (2000). Comparison between synchronous and asynchronous implementation of parallel genetic programming. In *Proceedings of the 5th International Conference for Artificial Life and Robotics*, pages 251–254, Japan.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, (59):433–460.

Ursem, R. (2002). Diversity-guided evolutionary algorithms. In Guervós, J. M. et al., editors, *Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 462–471, Granada, Spain. Springer.

Whigham, P. (1995). Grammatically-based genetic programming. In Rosca, J., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, CA, USA.

Whitley, D., Rana, S., and Heckendorn, R. (1997). Island model genetic algorithms and linearly separable problems. In Corne, D. and Shapiro, J., editors, *Proceedings of AISB Workshop on Evolutionary Computation*, volume 1305 of *LNCS*, pages 109–125, Manchester, UK. Springer.

Wineberg, M. and Oppacher, F. (2003). Distance between populations. In Cantú-Paz, E. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2724 of *LNCS*, pages 1481–1492, Chicago, IL, USA. Springer-Verlag.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wright, S. (1931). Evolution in mendelian populations. *Genetics*, 16:97–159.

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In Jones, D., editor, *Proceedings of the Sixth International Congress of Genetics 1*, pages 356–366.

Zhang, B.-T. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38.

Zhu, Z.-Y. and Leung, K.-S. (2002). Asynchronous self-adjustable island genetic algorithm for multi-objective optimization problems. In Fogel, D. et al., editors, *Proceedings of the Congress on Evolutionary Computation*, pages 837–842, Honolulu, USA. IEEE Press.