Population Variation in Canonical Tree-based Genetic Programming

Peyman Kouchakpour

PhD Thesis
School of Electrical, Electronic and Computer Engineering
University of Western Australia
Nedlands, Perth,
Western Australia

May, 2008

32 Hollister Way
Noranda
Western Australia, 6062

The Dean
Faculty of Engineering and Mathematical Sciences
The University of Western Australia
Nedlands
Western Australia, 6907

15th January 2008

Dear Professor Mark Bush,

This thesis entitled "Population Variation in Canonical Tree-based Genetic Programming" is submitted for the fulfilment of the requirements for the degree of Doctorate of Philosophy (PhD) at the University of Western Australia.

Sincerely yours,

Peyman Kouchakpour

In Memory of My Mother

Mehry Kouchakpour

"Only a life lived for others is a life worthwhile", Albert Einstein.

Regard man as a mine rich in gems of inestimable value. Education can, alone, cause it to reveal its treasures, and enable mankind to benefit therefrom.

Bahá'u'llah

Abstract

The Genetic Programming paradigm, which applies the Darwinian principle of evolution to hierarchical computer programs, has produced promising breakthroughs in various scientific and engineering applications. However, one of the main drawbacks of Genetic Programming has been the often large amount of computational effort required to solve complex problems. There have been various amounts of research conducted to devise innovative methods to improve the efficiency of Genetic Programming. This thesis has three main contributions. It firstly provides a comprehensive overview of the related work to improve the performance of Genetic Programming and classifies these various proposed approaches into categories. Secondly, a new static population variation scheme (PV) is proposed, whereby the size of the population is varied according to a predetermined schedule during the execution of the Genetic Programming system with the aim of reducing the computational effort with respect to that of Standard Genetic Programming. Within this new static scheme the initial population size is made to be different from the initial size of the Standard Genetic Programming such that the worst case computational effort is never greater than that of the Standard Genetic Programming. Various static schemes for altering population size under this proposal are investigated using a comprehensive range of standard problems to determine whether the nature of the "population variation", i.e. the way the population is varied during the search, has any significant impact on Genetic Programming performance. It is shown that these population variation schemes do have the capacity to provide solutions at a lower computational cost compared with the Standard Genetic Programming. Thirdly, three innovations for dynamically varying the population size during the run of the Genetic Programming system are proposed. These are related to what is called Dynamic Population Variation (DPV), where the size of the population is dynamically varied using a heuristic feedback mechanism during the execution of the Genetic Programming with the aim of reducing the computational effort. The efficacy of these innovations is examined using the same comprehensive range of standard representative problems. It is shown that these new ideas do have the capacity to provide solutions at a lower computational cost compared with standard genetic programming and previously reported algorithms. Finally, further interesting research potentials for population variation are identified together with some of the open areas of research within the Genetic Programming and also possible future trends in this discipline.

Acknowledgement

I would like to express my special thanks and gratitude to my supervisors Professor Anthony Zaknich and Professor Thomas Bräunl for their continual support, patience, guidance, inspiration and encouragement.

I would like to also thank my beautiful and beloved wife and lifelong partner, Nasim Kouchakpour, for her constant support, understanding and encouragement.

I would like to show my appreciation to my family and close friends for their support.

Last but not least, I am exceedingly grateful and forever indebted to my mother, Mehry Kouchakpour, and my father, Dr Arash Kouchakpour, for all the sacrifices they made all through my life and their continual support.

I am certain that Mum would have been extremely proud to see this.
I have dedicated this work to you, Mum.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND OVERVIEW

"The wisdom of the wise and the experience of the ages are perpetuated by quotations", Benjamin Disraeli.

1.1 Introduction to Evolutionary Algorithms

Problem solving, which is considered the most complex of all the intellectual functions, is a higher-order cognitive process that requires the control of more routine or fundamental skills [99]. This intricate and complicated task of problem solving has been facilitated by the advent of computers. The automatic computational solution of problems has been central to artificial intelligence (AI), machine learning, and the broad area encompassed by what Turing called "machine intelligence" [174]. In the organic world, the most complex and diverse organisms and species came into existence through the process of evolution. The human brain, the most powerful natural problem solver with some one hundred billion neurons and tens of thousands of other nerve cells with quadrillion (10^{15}) synaptic connections, was formed through the evolutionary process. Consequently, natural and biological processes have continually served as an inspiration, providing valuable sources of ideas and metaphors to researchers. In his 1859 paper, On the Origin of Species by Means of Natural Selection, Charles Darwin [55] introduced the theory and concept of natural selection and the phenomenon of survival of the fittest, which governs the evolutionary adaptation of the biological world from the smallest virus to the most complicated mammals. The abundance of incredibly complex and evidently intelligent natural creatures has led scientists to view evolution as a powerful paradigm. Based on the Darwinian principles of natural selection, researchers have drawn inspiration from the molecular genetics to validate whether the effects proven in the carbon world could possibly also occur in the silicon world. This resulted in the birth of Evolutionary Computation (EC) with its four main traditional variants, namely Evolution Strategies (ES), Evolutionary Programming (EP), Genetic Algorithms (GA) and Genetic Programming (GP).

In 1948, Turning proposed "genetical or evolutionary search" and during the 1960s and 1970s three different implementations of the basic idea were cultivated. Fogel, Owens and Walsh invented Evolutionary Programming (EP) [86][87] and Rechenberg and Schwefel developed Evolution Strategies (ES) [184][279], while John Holland promoted his method of the Genetic Algorithm (GA) [118][119]. During the 1990s a fourth stream materialised, Genetic Programming (GP) championed by Koza [170][174]. These various dialects of EC are designated as Evolutionary Algorithms (EA) and are distinguished by types of structures or representations comprising the individuals in the population, genetic operations, selection procedures and a host of other items. The representation in Evolutionary Programming (EP) is based on real-valued vectors with the only variation operator being mutation, based on Gaussian perturbation. EPs have a deterministic parent selection and are evolved in the form of Finite State Machines (FSM). The structures in Evolution Strategies (ES) are based on real-valued objective variables. Similarly, ES utilise mutation, based on Gaussian perturbation, but they are frequently associated with engineering continuous parameter optimisation problems. The representation in Genetic Algorithms (GA) is string based, being a string of binary digits, integer, real-valued or floating-point representations. GAs are the most widely known type of EA. The youngest member of Evolutionary Algorithms is Genetic programming (GP), with its particular representation of trees as chromosomes. Genetic Programming differs from the other EA disciplines in its application area. While the other Evolutionary Algorithms normally pertain to optimisation problems, Genetic Programming is related to machine learning. GP is utilised to discover systems (Figure 1.1a), whilst most other Evolutionary Algorithms generally seek for input to optimise the solution to the system (Figure 1.1b).

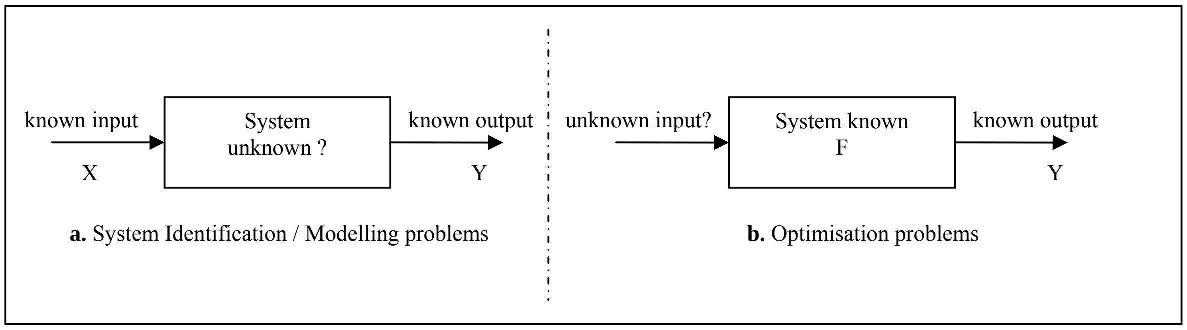


Figure 1.1 Modelling and optimisation problems

In EA, an initial population of individuals is randomly generated. Each individual is measured in terms of how it performs in solving a particular problem. The Darwinian principle of reproduction and survival of the fittest together with the genetic operations such as sexual recombination are used to create new offspring population of individuals from the current population. The environmental pressure causes natural selection, triggering a rise in the fitness of the population. This process is iterated until a candidate with a desired quality is realised or until a previously defined computational limit is reached. To illustrate the workings of typical EAs, consider a one dimensional objective function to be minimised. In other words, it is required to find a solution x^* on a set of possible solutions *S*, such that $x^* \neq x \Rightarrow f(x^*) \leq f(x)$ (inequality reversed for maximisation). Three stages of the evolutionary search are depicted in Figure 1.2, namely the distribution of the individuals directly after the initialisation exhibiting random spread over the solution space, the individuals after some generations displaying fitness improvement due to pressure exerted by natural selection and finally the individuals at the end of the evolution, where most of the individuals are concentrated around the troughs. This is an example of a multimodal problem with both a local optimum and a global optimum.

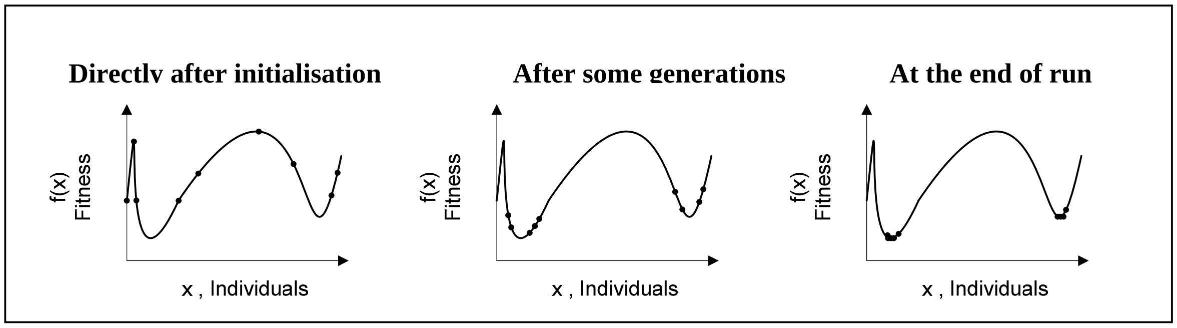


Figure 1.2 Population distribution – progress of an Evolutionary Algorithm

It is obvious that the problem complexity increases with the dimension. For example, Figure 1.3 illustrates a two dimensional maximisation problem. On this landscape the height dimension belongs to the fitness function and the other two dimensions represent biological traits, where the *x*-*y* plane holds all the possible trait combinations. The pinnacles of the landscape correspond to a range of successful trait combinations, while the low regions represent less fit combinations. Such a plot is a representation of the problem's search space. As it can be seen, the search space or fitness landscape can be of surprisingly complex topography. The process of gradually advancing the population to the peak areas, powered by natural selection and genetic variation, is called evolution. For more complex problems, with more than two parameters or variables, the situation becomes harder to visualise.

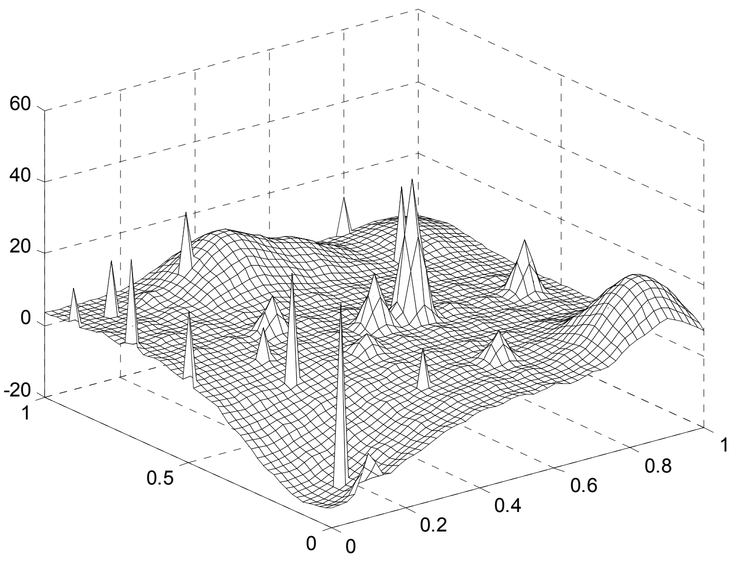


Figure 1.3 A complex search space of a two-dimensional maximisation problem

Although the main goal is the detection of the global optimum, there is a possibility that the algorithm may pinpoint and get trapped in one of the local optima. The search is normally categorised into two distinct phases of exploration versus exploitation, which concentrates the search in the vicinity of a good known solution. There are tradeoffs between the exploration and exploitation, where too much of the former can lead to inefficient search and too much of the latter can lead to a tendency to focus the search too early and thereby losing population diversity too quickly, which can result in premature convergence. The balance between exploring the search space and exploiting the discovered highly fit positions is a recurrent theme in the theory of Evolutionary Algorithms. The increased exploitation can lead to faster progress of the algorithm but at the same time, it can result in greater possibility of failure to spot the global optimum. Figure 1.4 portrays the development of the best-so-far individual, where the progress is rapid during the initial generations and then flattens out further on in the evolution. In this plot, the run is divided into two equally long sections *A* and *B*. The plot can suggest that very long runs may not be that beneficial and efforts spent after a certain number of fitness evaluations may not result in better solution quality [71].

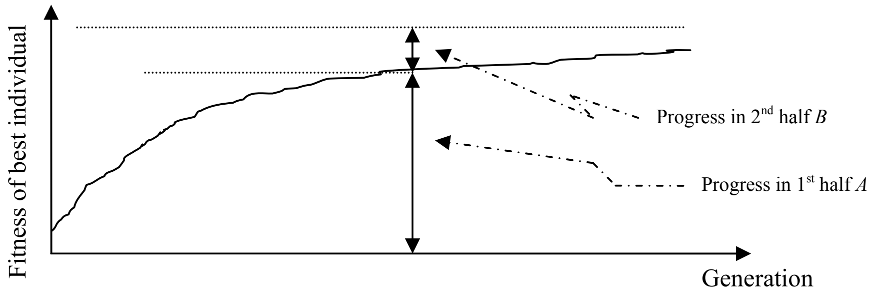


Figure 1.4 Progress of EA in terms of the best fitness value within population

Evolutionary Algorithms are generally considered to be robust techniques of almost universal application. This is due to the fact that they can be implemented on a wide variety of problems with minimal adaptation, but are likely to be much less efficient than the highly tailored problem-specific algorithms. This observation is evident in Figure 1.5, where EAs show a roughly evenly good performance for a wide range of problems, but are inferior to the problem-tailored algorithms that are specifically designed for the problem of interest [1][71].

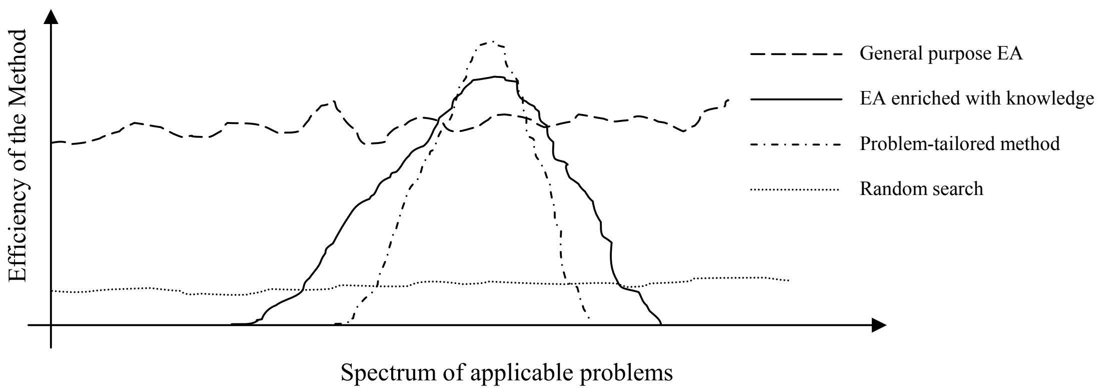


Figure 1.5 Comparison of robustness of EAs with more traditional practices

1.2 Components of Evolutionary Algorithms

In this section some of the main components of Evolutionary Algorithms are briefly described.

1.2.1 Definition of Individuals

In the discipline of molecular genetics, each individual is a dual entity: its phenotypic and genotypic properties. An individual's genotype encodes its phenotype. Hence, the genotype contains all the necessary information to build the particular phenotype. This concept of duality is also applied in the field of evolutionary algorithms. The process of mapping phenotypes from a phenotype space ***P*** onto the genotype space ***G*** using a defined rule F (F:$\mathbf{P} \rightarrow \mathbf{G}$), is called representation. It should be noted that the phenotype space ***P*** can be very different to the genotype space ***G***. For instance, the representation typically used in Genetic Algorithms is fixed length binary characters. For example, in an integer optimisation problem the phenotype *90* is encoded as a binary string of *1011010*. The whole evolutionary search takes place in the genotype space ***G***. This means that the final solution obtained by the evolutionary process is required to be decoded back into its phenotype space ***P***.

1.2.2 Population

Classic adaptive and machine learning paradigms generally analyse only a single solution point. For example, in Neural Networks (NN) a single vector of weights in the weight space undergoes adaptation, or in Hill Climbing a single point in the search space and in Simulated Annealing a single domain specific structure in the search space undergoes adaptation. However, in scatter search, ant systems, swarm intelligence and evolutionary algorithms etc, a population or a set of solution points undergo adaptation simultaneously. The population, which is a collection of a number of individuals from the genotype space ***G***, forms the unit of evolution. The first population is randomly generated using a stochastic process, commonly known as initialisation.

1.2.3 Fitness Function

The fitness function is an expression of the environmental requirements and an estimation of solution quality. The fitness function basically assigns a scalar fitness value or a quality measure q to the genotype ψ ($q(\psi)$, $\psi \in \boldsymbol{G}$) by means of some well-defined evaluative procedure according to how well the individual attempts to solve the particular problem at hand. The search process, which is driven by the variation and selection operators, is highly dependent on the fitness function. The fitness function is the driving force of evolutionary algorithms.

1.2.4 Selection Mechanism

The individuals within the current population are selected based on their fitness to participate either for mating or replacement, forming the new individuals for the next generation. The role of survivor selection or replacement is to select individuals, favouring those with higher quality, from the current population and copy it unaltered into the new population. The role of mating or parent selection is to allow better individuals to become parents. The selected candidates undergo variation creating offspring for the new population.

1.2.5 Variation Operators

The variation operators are ultimately used in steering the evolutionary process into new unexposed territories within the search space. The role of the variation operators is to generate new candidate solutions. Variation operators are commonly divided based on their arity.

Definition 1.1 The **arity** is the number of arguments or inputs that a function can take.

The most well known unary variation operator is the mutation operator applied to one genotype, producing slightly modified mutant. It randomly modifies the genetic material of one parent, creating a new offspring or child. The most widely used binary variation operator is the recombination or crossover operator, which merges the content of two parent genotypes into either one or two offspring genotypes.

1.2.6 Termination

The evolutionary process is stopped when specific termination criteria are met. For problems with known fitness level or a known optimum O ($O = \max\{q(\psi) \mid \psi \in \boldsymbol{G}\}$), the stopping condition is generally reaching this level within a precision of $\varepsilon > 0$. In other instances, the process can be stopped when a certain condition X is reached. This condition can be the maximum allowable CPU time or the total number of fitness evaluations reaching a certain predefined limit.

1.3 Overview of Tree-based Genetic Programming

The genetic programming paradigm applies the Darwinian principle of evolution to hierarchical computer programs of dynamically varying shapes and sizes. Figure 1.6 shows a simplified contour plot of a possible solution space. The programs are represented as dots (square, triangular or parallelogram) representing individuals at different stages of evolution, e.g. initial individuals shown as square, generation 10 as triangle and generation 40 as parallelogram. Darker areas represent fitter landscapes. GP solves problems by searching the space of all possible computer programs, composed of functions and terminals suitable for the problem domain. The functions may be standard programming functions, arithmetic operators or mathematical functions, logical functions or any domain specific functions. The population of thousands of computer programs are genetically bred using the Darwinian principle of natural selection and genetic operations. An initial population of computer programs is randomly generated. Each individual computer program within this population is then measured using a predefined problem specific fitness function. For example, if the problem at hand involves creating a randomiser, the fitness of the programs can be assessed via entropy, with higher entropy indicating a superior randomiser. The fitness measure provides an indication of how well a computer program performs in solving problems in a given problem environment.

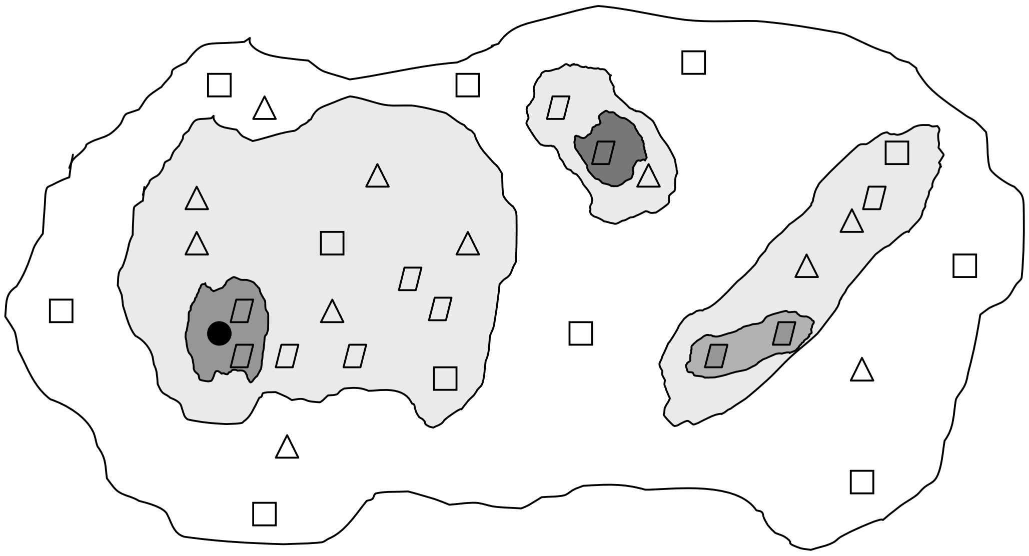


Figure 1.6 Simplified contour plot of a solution space

Each individual program is typically tested with a number of fitness cases representing different environments or situations. For instance, multiple fitness cases can represent different initial conditions or a sampling of values of an independent variable. The fitness of the individual program can be measured in terms of the difference between the output produced by the program and the correct answer to the problem. This class of error measure ρ can be based on the Minkowski norm defined in a Euclidean space, as defined by the general equation (1.1).

$$\rho_r(\vec{d},\vec{y}) = \frac{1}{B}\sum_{j=1}^{B}(|d_j - y_j|)^r \qquad (1.1)$$

,where

$\vec{d}$	is the desired vector output or actual solution vector to the problem
$\vec{y}$	is the actual vector output produced by the program
d_j	is the j^{th} dimensional element of vector $\vec{d}$
y_j	is the j^{th} dimensional element of vector $\vec{y}$
B	is the vector dimension or number of fitness cases
r	is a positive integer in the range of $[1,\infty)$.

The error measure ρ is the average absolute error for $r = 1$, the mean square error for $r = 2$ and the maximum absolute error $\lim_{r\to\infty}[\rho_r(\vec{d},\vec{y})]^{1/r}$ for $r \to \infty$. The most commonly used error measures are the Mean Square Error (MSE) and the average absolute error. New offspring of computer programs are created from the current population using the Darwinian principle of survival and reproduction of the fittest and the genetic operations. The actual selection of computer programs for reproduction or recombination is probabilistic but favours those with high fitness measure. The offspring programs contain building blocks or subroutines from their parents. Selected arbitrarily chosen parts of effective programs are randomly recombined to possibly produce new fitter computer programs. The new population of individual computer programs replaces the old existing population. The fitness of each of these new individuals is then measured and this process is repeated over many generations until a suitable candidate program evolves or some termination criterion is met. The process of the classical genetic programming paradigm is summarised in a flowchart as depicted in Figure 1.7 with its simplified pictorial representation of the GP cycle in Figure 1.8. The index *i* refers to an individual in the population of size *M* with variable *GEN* indicating the current generation number. The best individual which has appeared in any generation (best-so-far individual) is designated as the result of the GP process. It should be noted that the structures, which undergo adaptation, are active and not passive encodings of the solution to the problem. Hence, they are capable of being executed in their current form.

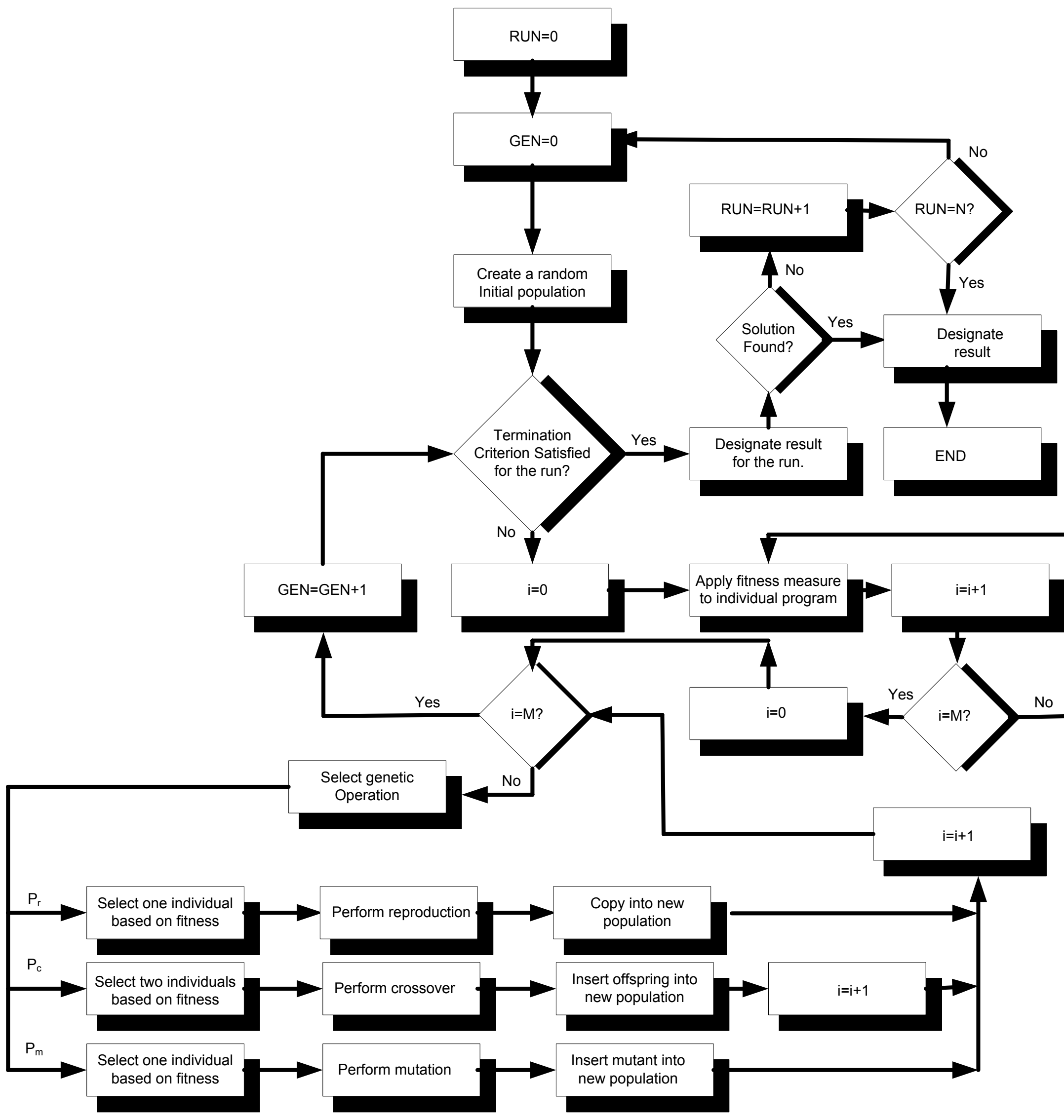

RUN=0
GEN=0
No
RUN=RUN+1
RUN=N?
No
Yes
Create a random Initial population
Solution Found?
Yes
Designate result
Termination Criterion Satisfied for the run?
Yes
Designate result for the run.
END
No
GEN=GEN+1
i=0
Apply fitness measure to individual program
i=i+1
Yes
i=M?
i=0
Yes
i=M?
No
Select genetic Operation
No
i=i+1
P_r
Select one individual based on fitness
Perform reproduction
Copy into new population
P_c
Select two individuals based on fitness
Perform crossover
Insert offspring into new population
i=i+1
P_m
Select one individual based on fitness
Perform mutation
Insert mutant into new population

Figure 1.7 Flowchart of classical genetic programming paradigm

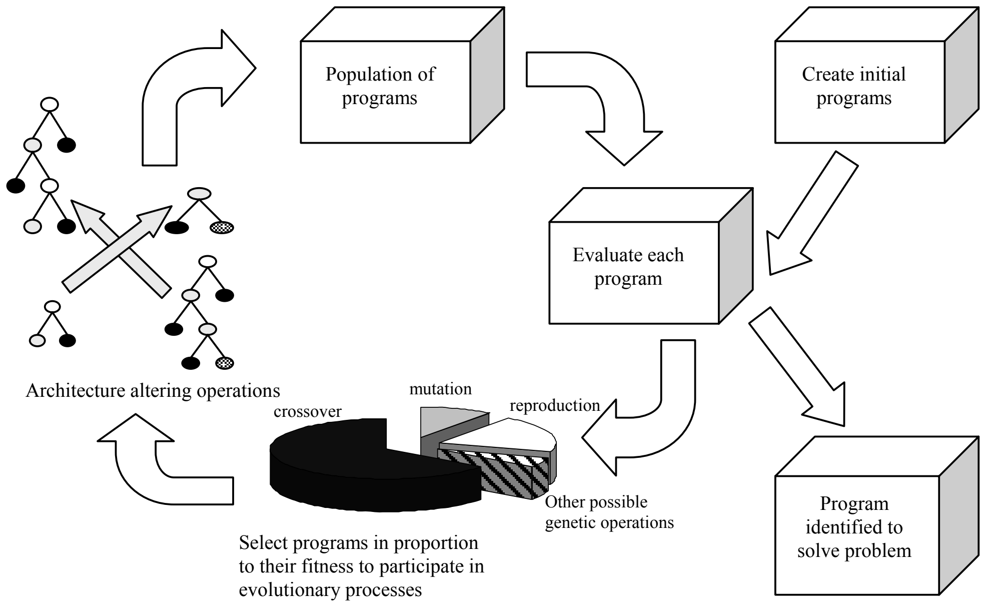

Population of programs
Create initial programs
Evaluate each program
Program identified to solve problem
Architecture altering operations
mutation
crossover
reproduction
Other possible genetic operations
Select programs in proportion to their fitness to participate in evolutionary processes

Figure 1.8 The simplified GP cycle

It should be noted that research in the fields of machine learning, artificial intelligence etc. concentrate on methods that are correct, consistent, justifiable and deterministic. Although these principles are valuable and play significant roles in most branches of sciences, mathematics and engineering forming an integral part of our thinking, they are not necessarily applicable to genetic programming. For example, consider a GP solving the problem applicable to the general first order differential equation of the form $y' + \phi(x)y = \zeta(x)$, where y represents the dependant variable and $\phi(x)$ and $\zeta(x)$ represent two different continuous functions of the independent variable *x*. It is quite possible that the GP may produce a result indicated by (1.2), where *C* is the constant of integration.

$$y = e^{-\int \phi(x)dx}\left(\int e^{\int \phi(x)dx}\zeta(x)dx + C\right) + 10^{-20}\xi(x)\phi(x) \qquad (1.2)$$

Although the last term of (1.2) may be quite small and negligible from a real-life engineering problem point of view, it would simply be not acceptable for most scientists or engineers. Therefore, the above solution is not a correct solution to the first order DE. The extra right side term in (1.2) has no justification and there is no logical sequence of reasoning for this term based on the mathematics of differential equations. In addition, GP promotes contradictory and inconsistent solutions. A new run of GP on the above problem can yield a completely different solution. The genetic programming paradigm is a highly stochastic process. In fact, its probabilistic nature is one of its key factors in solving problems with the notion of "anything can happen and nothing is guaranteed" [170]. Although the above stated principles are violated in GP, during the period of its infancy Koza showed that GP can be successfully applied to a wide variety of problems from many disciplines, such as the artificial ant problem, symbolic regression problem, Boolean multiplexer problem, Boolean parity function, discovery of trigonometric identities, symbolic integration and differentiation, sequence induction, numeric roots of equations, programmatic image compression, wall following robot, box moving robot, randomisers, the Boolean Symmetry problem, the lawnmower problem, finding an impulse response function, the obstacle avoiding robot, the minesweeper problem, discovery of detectors for letter recognition, predictions of transmembrane domains in proteins and prediction of omega loops in proteins to name but a few [170][174]. More recently, Koza [182][184] has shown real-life applications of GP in performing automated synthesis of controllers, automated synthesis of analog electrical circuits such as Butterworth or Chebychev filters, amplifiers or MOSFET circuits, antennas, genetic networks and metabolic pathways. It should be noted that GP's problem representation is a superset of the representations of all of the other machine learning (ML) paradigms [21]. More importantly, GP can deliver human competitive machine intelligence [184] and can be used as an automated invention machine. Specifically, it was argued [184] that GP had duplicated the functionality of fifteen previously patented 20th century inventions, six previously patented 21st century inventions and has produced two instances of patentable new inventions. For a more detailed description of genetic programming the reader is referred to Chapter 3.

1.4 A Brief Literature Survey in GP

This subsection summarises some of the previous research conducted in the field of Genetic Programming. The reader, not familiar with Genetic Algorithms (GA) and/or Genetic Programming (GP), is encouraged to read the next two chapters on GA and GP before reading this section.

Research within the Genetic Programming community has been divided into three main areas. One area of GP research has concentrated on the applications of GP and extending its innovative applications to real-world problems. Another area of focus has been on enhancing our understanding of fundamental aspects of genetic programming and examining how and why it works. A further area of research has given attention to discovering new techniques, which could increase the power and performance of the genetic programming approach. Although some parts of this subsection briefly discuss the various techniques that have been introduced to improve the performance of GP, a whole chapter is devoted to this topic in Chapter 5, where an attempt has been made to categorise all the suggested techniques.

A wide range of applications of genetic programming have been studied. The problems themselves could be separated into artificial problems and real-world or scientific problems such as engineering and medical specific related problems. Applications on the artificial problems have been widespread from applications in games [69] to simulated problems such as the sort problem [159]. Genetic Programming was used to solve the artificial ant problem together with the pursuer and evader game [168]. In [171], GP was applied to two well-known benchmark problems from the field of neural networks (NN), namely the truck backer problem and the inter-twined spirals problem, which is a challenging classification problem. GP was employed to teach a computer to develop strategies for the ancient Egyptian board-game Senet [84]. GP has been applied in various branches of engineering and sciences from neural networks [1] [56] [169] [333], image processing and pattern recognition [51] [124] [172] [267], biomedical science [60] [100] [102] [181] [281] to control engineering [135] [144] [220] and robotics [42] [110] [198] [199] [221] [225] [229] [314]. In the biomedical application field such as molecular and biochemical genetics and biomedical engineering, GP was applied to pattern recognition analysis from the spectra of human tumour biopsy extracts [222]. The detection of epileptic events was investigated in [219] , whereas [175] [176] applied GP to recognise a given protein segment as being a transmembrane domain or non-transmembrane domain area of the protein. Circuit design [1] has been another focus in the applications of GP. In [66] [111], the synthesis of digital logic expressions was illustrated while [178] engaged in electronic circuit synthesis, designing a low-pass filter, an amplifier and an asymmetric band-pass filter. Multiple circuits such as the time-optimal controller circuit, the temperature-sensing circuit and the voltage reference circuit were designed using GP [179]. Koza *et al*. [180] designed analog circuits that perform mathematical functions such as cube and square root, for which no circuit has been found in the published literature. GP has made some penetration into the area of image processing. In [1], rules for optical character recognition were evolved and upgraded. In [302], GP was applied to learning algorithms that investigate face images, which are coordinated into a face recognition system. There have been many applications in traditional engineering such as electrical engineering. In [173], the impulse response function for a linear time-invariant system was derived while Sharman *et al*. [282] evolved the structure and parameters of adaptive digital signal processing (DSP) algorithms. Automatic generation of neural network architecture was carried out in [310]. GP has made some inroads into robotics. A complete robot system including its controller and physical structure were synthesised and designed [200]. Uchibe *et al.* [309] discussed how multiple robots could engage in cooperative behaviour using co-evolutionary processes, while the synthesis of the design of both the topology and parameter values for controllers was automated through the use of GP [183]. Tunstel *et al.* [307] examined three soft computing paradigms, namely GP, GA and neural networks, for automated learning in robotic systems. Kurashige *et al.* [187] adopted GP as a learning method to construct the motion planning system for a six legged locomotion robot while [248] reported on evolving bipedal locomotion using GP. There have been many other instances and other disciplines where GP has been applied, such as application of GP to information retrieval systems [185], prediction of financial data [133] [147], software engineering [204] [318], classification tasks [306] [322] and navigation tasks [24], to name but a few.

Most of the research that has been carried out in genetic programming has looked into applying the genetic programming paradigm to problems. The majority of papers and books published have studied or illustrated applications to various disciplines. Incredibly little research has been devoted to the theory on the fundamental aspects of genetic programming. Also relatively little research has focused on performance improvement of Genetic Programming.

Although the amount of research on the fundamental aspects of GP and examining how and why it works is limited, the following efforts could be briefly summarised. Schema theorems are traditionally used to explain how genetic algorithms (GA) and, more recently, genetic programming (GP) work [264]. Conventional wisdom in GP has suggested what makes a problem difficult is a problem's domain [52]. The fitness landscape such as a rugged or deceptive fitness landscape could make a problem GP-Hard [52]. There are but a few theoretical works that address problem difficulty in GP at all. The first work to do so appeared in [170], where a semi-empirical formula was provided that estimated the number of trials needed to solve a problem with a specified success probability. The number of independent runs $R(z)$ with a success probability of z required to solve a specific problem by generation i is defined by (1.3).

$$R(z) = \left\lceil \frac{\log(1-z)}{\log(1-P(M,i))} \right\rceil \tag{1.3}$$

$P(M,i)$ is the cumulative probability of satisfying the success predicate of a problem for GP for generations between generation 0 and generation i. The number of runs required to yield a success with a probability of 0.99 (z = 99%) is depicted in Figure 1.9. Kinnear [160] suggested that the analysis of the structure of the fitness landscape might allow relating the difficulty of the problem to the structure of the landscape.

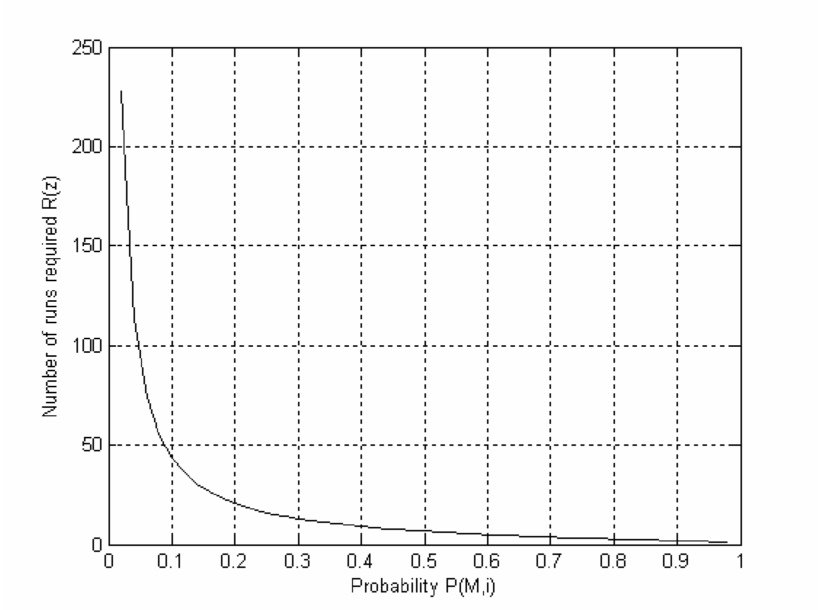


Figure 1.9 Number of runs required as a function of cumulative probability for z = 99%

The tendency of tree sizes to grow rapidly during GP is well known, causing excessive use of CPU time and memory. It was reported in [59] that tree size increases progressively as a function of the number of generations. When tree size starts to grow rapidly, a GP run almost always stagnates [27] and it is useful to take some measures against this. With multi-objective optimisation algorithms it is possible to optimise towards several objectives. There are only a few studies which perform multi-objective optimisation in the context of GP [27].

Definition 1.2 The **depth** of a node is the minimum number of nodes that is required to be traversed from the root node of the tree to the concerned node.

A limit on tree depth or number of nodes is normally manually set. However, setting a reasonable limit is difficult [27] and it is recommended to introduce program size as a second independent objective besides the program functionality. If the limit is too low, GP might not find a solution and if the limit is too high, evolution may slow down. In standard GP F_i is the fitness of individual i defined as the error E_i of an individual's output compared to the correct solution, i.e. $F_i = E_i$. A multi-objective GP was introduced in [27], reducing bloat by penalizing larger programs, adding a size dependant term to the fitness. If N_i is the number of edges of an individual and "α" is the parsimony factor, then:

$$F_i = E_i + \alpha \times N_i \tag{1.4}$$

However, parsimony pressure can drive the entire population to a minimal possible size and possibly lower the probability of finding good solutions. Another alternative is to optimise the functionality first then follow by optimizing the size. The population is divided into two groups. Group one contains individuals that have not reached an error smaller than ε, which is the maximum acceptable error. The fitness is calculated according to the error without any pressure on size, i.e. $F_i = E_i + 1$, if $E_i > \varepsilon$. Group two individuals are already within the acceptable error ($E_i < \varepsilon$) and hence their fitness will incorporate the size factor, in other words, $F_i = 1 - \frac{1}{N_i}$. In this way the fitness of an individual with a large tree size is nearly one while the fitness of a small tree size is much closer to zero. This will not hinder GP from finding good solutions as no pressure is applied unless the individual has reached the aspired performance.

Many researches have described and discussed the idea of introns [13] [134]. Several researchers have hypothesised that the growth of code during GP and the creation of non-functional code are both important contributors to the production of effective programs. Langdon [190] described that bloat is not specific to Genetic Programming. It is inherent in search techniques in any discrete variable length representation using simple static evaluation functions, provided there is no length bias. Igel *et al.* [134] described intron as excess size of a language expression when compared with a smaller equivalent expression (see Figure 1.10) and argued that an intron is a code that can be removed from a program without affecting its behaviour or without affecting the fitness of GP. For example, some introns are dead code, as they are never executed, or a code that does nothing.

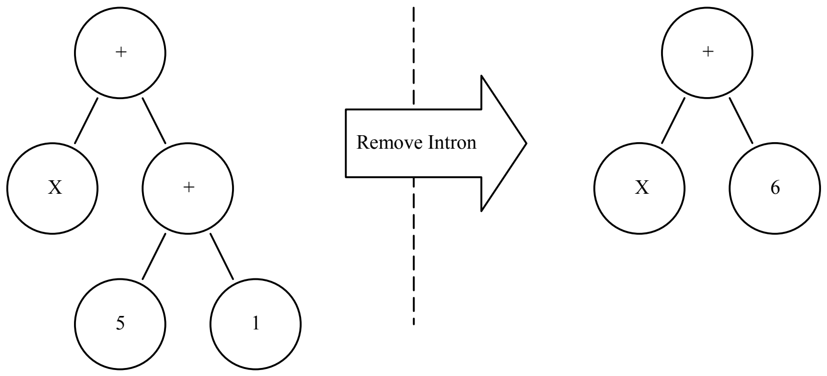


Figure 1.10 Illustration of Intron

Some researchers think that programs containing non-coding segments are protected from destructive effects of crossover and these segments promote the grouping of building blocks [22] [28] [242] [243]. Other authors [6] [109] [115] [209] [303] argue that useless code produces negative effects, exhausting memory space in individuals and altering the search process. Banzhaf [22] discussed two hypotheses, namely “fitness causes bloat” and “neutral code is protective”. Neutral code constitutes a major percentage of individuals. He argued that a general consensus has emerged in the GP community that the two mentioned phenomena of code growth and neutral code appearance are strongly related and that at this point it seems too early to draw conclusions as to the relative strength of their effects. A new implementation of introns was developed in [38], named evolutive introns (EIs). EIs had no effect on the depth of the individuals, and favoured the existence and growth of NOP (No Operations) chains. NOP chains appeared at any point in the trees, when chosen for crossover, which produced good offspring. This method consisted of adapting the probability of selection of every node in the tree after each application of the crossover operator, so that good building blocks were harder to disrupt. To avoid bloating, NOP instructions will not appear explicitly but implicitly; by associating each node in the tree with an integer (weight) (refer to Figure 1.11).

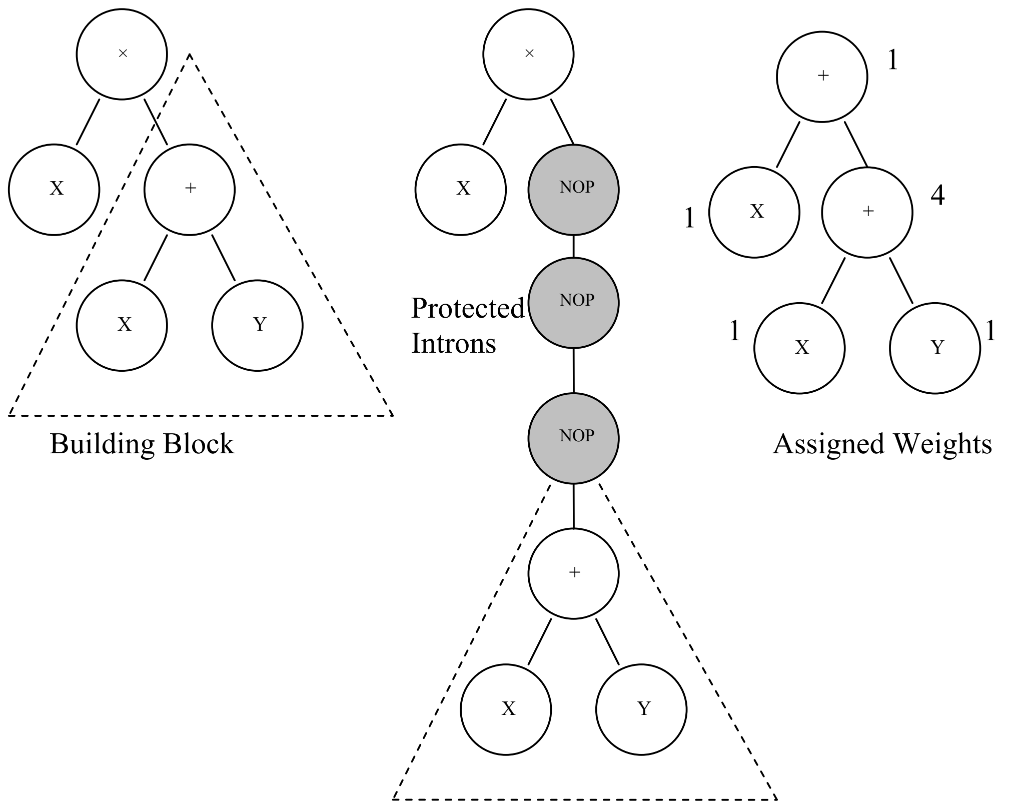


Figure 1.11 Tree with Evolutive Introns-NOP not coded, each node is associated with a weight

On the other hand, other authors [6] point out that the use of introns can degrade the search performance of GP because of the existence of useless code portions that have no effect in fitness calculation, and in some cases occupy all the available space for the tree representation of the individuals. Luke [209] argued that tree bloat is unnecessary. Hirasawa [115] blamed bloat for the difficulty to search an optimum solution in the GP search space of solutions. The searching efficiency of GP is not so high in some cases because the search space becomes enormous due to its bloat. Many researches have attempted to control and fight the bloat phenomenon. In fact, Terrio and Heywood [303] introduced directing crossover for reduction of bloat, where the fitness of each individual is measured and recorded at each individual node. The node whose fitness exceeds all the others is tagged as being the "index" node used as crossover points. In other words, directed crossover always takes place at the index node. Soule *et al.* [288] argued that programs will grow indefinitely, mostly via the increases in the amount of non-functional code, regardless of whether or not the growth acts to improve the program solution. It was argued that looking at biological evolutionary processes was also not very enlightening. In humans, roughly 80-90% of DNA (deoxyribonucleic acid) does not code for functional proteins, although some of this DNA does have a structural function [288]. Similar percentages hold for most other higher organisms. Despite the prevalence of non-functional code, removing this code at every generation does not halt the program's growth. It is believed that much of program's growth is not caused by pressure to improve the solution, but instead is an innate part of the GP process. Some research has attempted to pinpoint which parameters could impact on the performance of GP. Kinnear [159] highlighted that the number of individuals *M* in a population or the actual fitness cases can impact on the success of the GP. Handley [102] indicated that if *M* is too small, then GP could reduce to a random search while if *M* is too large, computation is wasted. He also argued that there is no theory to guide the choice of population size. Other researchers have looked into new ways and methodologies to improve GP performance. Methods using a subroutine such as ADF (Automatic Defined Functions) or MA (Module acquisition) have been developed, for an efficient way to search in GP. ADF were examined in-depth in Koza [174]. Koza *et al.* [173] demonstrated the value of ADFs in solving an impulse response function. Iba *et al.* [127] stated that traditional GP blindly combines sub-trees by crossover, which can disrupt beneficial building blocks. They believed that ADF can help to maintain useful building blocks. They stated that trees can grow exponentially large and suggested a method of pruning redundant trees. Crossover is destructive and methods such as ADF and MA have the advantage that a partial structure of a solution can be preserved from a destructive crossover [150]. The effects on performance of the primary GP operations for modification of structures have also been examined. Many researchers have attempted to highlight their importance. Crossover swaps randomly selected sub-trees between parents. Often the selection of a sub-tree is biased against the selection of a leaf. Koza [170] suggested a leaf selection frequency/probability of 0.1. As about 50% of the sub-trees are leaf nodes, it was determined [170] that it was important to bias selection of crossover points away from leaves and towards internal nodes. "This distribution promotes the recombination of larger structures whereas a uniform probability distribution over all points would do an inordinate amount of mere swapping of terminals from tree to tree in a manner more akin to point mutation than to recombination of small structures or building blocks". While the need to bias sub-tree crossover away from selecting leaves seems plausible it is not empirically validated. Some experiments have suggested that 0.1 may not be optimal for many problems [14]. Angeline [14] investigated the sensitivity of genetic programs that use sub-tree crossover to various values for the leaf selection frequency and showed that the optimal leaf selection frequency during the crossover is problem dependant. Most generally an effective method for determining appropriate leaf selection frequency may be a simple pseudo-adaptive random process. Beasley [23] argued that crossover is more important than mutation for rapidly exploring the search space and should be given a higher probability to alter genes. The effects of mutation, permutation, editing and encapsulation were studied in [170] and it was shown that there is no substantial difference in performance between the probability of success with and without these secondary genetic operations for the 6-Multiplexer problem. In fact, the mutation operator was not used at all in [174].

Hansen [104] found that crossover had a negative effect on the fitness of the offspring. The worth of a code fragment depends on the context within which it is executed, i.e. most often location, size, or function, or some combination thereof. Hansen argued that the insertion into a different program at a random location may destroy this context. Consequently, several researchers had proposed context preserving crossovers, with the aim of increasing the likelihood of moving the code fragment to a syntactically similar part of the recipient program in order to preserve its context and worth. Non-homologous crossover occurs when instruction blocks are exchanged between two evolved programs with no reference to the size and location of the two sets of instruction blocks. Hansen [104] proposed a homologous crossover attempting to mimic natural evolution by exchanging sets of contiguous instruction blocks between the two evolved programs. The groups of contiguous instruction blocks were selected such that the sets from each parent program were the same length and were taken from the same position in both of the two parent evolved programs. With uniform crossover, copies of the parent trees are recursively traversed starting from the root nodes to identify a common region. Nodes in the common parts are swapped with uniform probability. In summary, homologous crossover is a method of recombination between equal length program fragments in the same positions in each parent. Korenaga [162] looked at algorithms to produce shorter programs and avoid premature convergence. Their algorithm included an elastic artificial selection, which selected the best and worst individuals and then stored them for predetermined generations. They considered two populations, namely the evolution population and storage population. To keep the variety of individuals, the individuals were reintroduced by replacing the same number of worst individuals at a reintroduction rate defined as the ratio of storage population to evolution population. They believed that in this way they could store useful schemata which are not necessary at this time but might be effective after some generations. Some researchers have looked into alternative representations to improve performance.

The genetic programming literature has consistently cited the importance of maintaining diversity as being crucial in avoiding premature convergence toward local optima as noted in several diversity studies (see [90]). Typical evolutionary algorithms contain a phase of exploration followed by exploitation. However, the type and amount of diversity required at different evolutionary times remains rather unclear [34].

The above is a very brief summary of some of the work carried out within the GP community. It does not in any way document all the efforts that have been conducted within this discipline. It should however be noted that most of the research has been carried out in applying the GP paradigm to applications, rather than looking at ways to improve GP performance and understanding the fundamentals of GP or developing a theory for GP. Chapter 5 summarises in-depth the various techniques proposed in the literature to improve the performance of GP and categorises these suggested techniques.

1.5 Scope of Work

It was indicated in [170] that not all the runs in GP are successful. In fact, it was pointed out that more will result in failure than success, e.g. in some examples a success of 16% was reported. Therefore, it was recommended in [174] to investigate ways for enhancing the performance of GP. A thorough analysis of some parameters that impact on the performance of genetic programming was conducted in [174], but further work was strongly encouraged: "The question is not definitely answered here; however, the results of these limited experiments may suggest further work (experimental and theoretical) that might lead to general conclusion". It was shown that the addition of extraneous variables to the terminal set dramatically degraded the performance of the cubic polynomial symbolic regression problem. Similarly extraneous functions in the function set degraded performance, except for the 6-Multiplexer problem, where there was improvement to performance when the *IF* function was added to the set. The effect of one extraneous turning function was reported as negligible for the artificial ant problem. No substantial difference in performance was observed when extraneous ephermal random floating-point constants were introduced for the quadratic regression problem. Table 1.1 summarises these findings.

Parameters	Problem	Performance Effect
Extraneous Variables	Cubic Regression	Dramatically degrades
Extraneous Random Constants	Quadratic Regression	No change
Extraneous Functions	6-Multiplexer Problem	Improves Performance
Extraneous Functions	Artificial Ant Problem	Negligible – No change

Table 1.1 Extraneous variables, functions and constants and their impact

The success of the three different methods of *FULL*, *GROW* and *RAMPed half-and-half* were also investigated for the quadratic regression problem, 6-Multiplexer problem, linear equation problem with two unknowns and the artificial ant with the Santa Fe Trail. As illustrated in the Table 1.2 the *RAMPed half-and-half* was shown to be the best observed for these problems.

Problem	Probability of success		
	Full	Grow	Ramped half-and-half
Quadratic Regression Problem	3%	17%	23%
6-Multiplexer Problem	42%	53%	66%
Artificial Ant – Santa Fe Trail	14%	50%	46%
Linear equations with two unknowns	6%	37%	53%

Table 1.2 Comparison between full, grow and ramped half-and-half

It was also argued that there are a large number of operational questions surrounding the use of GP for which no definitive answer is known [170].

As it was mentioned in the previous subsection, most of the research within the GP community has concentrated its efforts on applying the GP paradigm to applications, rather than looking at ways to improve its performance. The majority of papers and books published have studied or illustrated applications to various disciplines. Little research has focused on performance improvement of Genetic Programming. In this thesis a comprehensive survey is firstly conducted on the current schemes proposed to improve the performance of GP with all these improvements methodically categorised. More importantly, it is believed that population variation can significantly enhance the performance of GP. In addition to the assessment and taxonomy of proposed schemes, the main aim of this thesis was to carry out an investigation of possible impacts on population variation with reference to performance of the genetic programming paradigm. A new static population variation scheme (PV) is proposed, whereby the size of the population is varied according to a predetermined schedule during the execution of the Genetic Programming system with the aim of reducing the computational effort with respect to that of Standard Genetic Programming. Within this new static scheme the initial population size is made to be different from the initial size of the Standard Genetic Programming such that the worst case computational effort is never greater than that of the Standard Genetic Programming. Various static schemes for altering population size under this proposal are investigated using a comprehensive range of standard problems to determine whether the nature of the "population variation", i.e. the way the population is varied during the search, has any significant impact on Genetic Programming performance. It is shown that these population variation schemes do have the capacity to provide solutions at a lower computational cost compared with the Standard Genetic Programming. Finally, three innovations for dynamically varying the population size during the run of the Genetic Programming system are proposed. These are related to what is called Dynamic Population Variation (DPV), where the size of the population is dynamically varied using a heuristic feedback mechanism during the execution of the Genetic Programming with the aim of reducing the computational effort. The efficacy of these innovations is examined using the same comprehensive range of standard representative problems. It is shown that these new ideas do have the capacity to provide solutions at a lower computational cost compared with standard genetic programming and previously reported algorithms.

1.6 Chapter Overview

In this chapter the field of evolutionary computing was briefly introduced. The main components of the evolutionary algorithm were presented. An overview of genetic programming was discussed and a brief literature survey was given. This chapter concludes with the scope of work for this thesis. Since GP is an extension of the conventional GA, in the next chapter the genetic algorithm, the most widely known type of evolutionary algorithm initially conceived by Holland, is reviewed. It also covers the theoretical aspects of GA. Chapter three embarks on an in-depth discussion of the theory of genetic programming. Chapter four introduces some of the theoretical aspects of genetic programming. Chapter five presents a survey of all the previous work conducted in GP research to improve its performance and categorises these techniques. Population variation is introduced in chapter six with the exploration and investigation of dynamic population variation in chapter seven. The analysis and evaluation of the proposed schemes is conducted in chapter eight and finally the conclusion and future research directions are offered in chapter nine.

CHAPTER 2

INTRODUCTION TO GENETIC ALGORITHMS

“A tree as big as you can reach starts with a small seed, a thousand-mile journey starts with one small step”, Lao-tse.

2.1 Introductory Example

A very simple example is presented here to introduce the basic concepts of the genetic algorithms. It should be noted that this example will incorporate the methodologies or operations of the GA in a very simplistic fashion to only convey the working concepts of the GA. This example contains an optimisation problem in finding the best business strategy for the fabrication of a particular solid state device that will offer better performance at a lower cost per unit. The business strategy involves making three decisions, namely the material to be used, the structure or the specific spatial arrangement of atoms required within the selected material and the fabrication technique used. These are further defined as follows:

- **Material**: Should an elemental semiconductor such as silicon (Si) be used or a compound such as cadmium telluride (CdTe)?
- **Structure**: Should the selected material be amorphous or crystalline?
- **Fabrication Technique**: Should an abrupt junction be formed by using a carrier gas of silicon tetrachloride ($SiCl_4$) and hydrogen (H) with dopant impurity atoms of diborane (B_2H_6) in an epitaxial growth process or graded junction be produced using thermal diffusion of phosphine (PH_3) over the surface of the semiconductor wafer?

The goal is to determine the best combination of these three choices that will yield a high performing electronic device at the lowest possible cost. The first step to solve this problem entails the identification of a suitable representation scheme. All the possible business strategies of these three decision variables can be successfully represented using a 3-bit binary encoding. Table 2.1 shows a random initial population of four genotypes, the corresponding phenotypes and their respective fitness values.

String no *i*	**Material**	**Structure**	**Fabrication Technique**	**Binary representation**	**Fitness $f(X_i)$**
1	CdTe	crystalline	epitaxial growth	011	3
2	CdTe	amorphous	thermal diffusion	000	0
3	Si	amorphous	epitaxial growth	101	5
4	CdTe	crystalline	thermal diffusion	010	2

Table 2.1 Representation scheme for the fabrication of a particular solid state device

Each individual in the population is tested using a payoff or fitness function for each generation to ascertain its fitness. Table 2.1 shows the fitness for the initial random generation ($g = 0$, where g is the generation number) of four individuals ($M = 4$, where M is the population size). The reader will probably notice that for simplicity the fitness of each string has been made equal to the decimal equivalent of the binary chromosome. The genetic algorithm is a highly parallel mathematical algorithm which uses the fitness values associated with each individual (typically fixed-length binary character strings) to transform individuals from the current set (called a population) to a new population of offspring objects using operations based on the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operations, such as crossover (sexual recombination) and mutation. Based on fitness, some of the better candidates are chosen probabilistically to seed the next generation for applying recombination and reproduction, forming the mating pool. The probability of each individual $P(X_i)$ $i \in \{1,2,3,4\}$ can for example be calculated using a useful analogy of the roulette wheel, where each individual will occupy a sector of an area proportional to its fitness within the wheel. Table 2.2 shows the mating pool and next generation (generation 1) individuals.

	Generation 0				**Generation 1**		
i	**String X_i**	**Fitness $f(X_i)$**	**$P(X_i)$**	**Mating Pool**	**Operation**	**String X_i**	**Fitness $f(X_i)$**
1	011	3	0.3	011	Mutation	010	2
2	000	0	0	101	Reproduction	101	5
3	101	5	0.5	101	Crossover	110	6
4	010	2	0.2	010	Crossover	001	1
Total		10					14
Worst		0					1
Average		2.5					3.5
Best		5					6

Table 2.2 One possible mating pool and outcome of applying genetic operations to create generation 1

The environmental pressure causes natural selection triggering a rise in the fitness of the population. This can be seen in Table 2.2 as the *best-of-generation* individual has a fitness of 5 and the *worst-of-generation* individual has a fitness of 0 for the initial generation, whereas their individual fitnesses have improved to 6 and 1 respectively for $g = 1$. In addition, the sum of fitness values of all individuals has now improved from 10 to 14 and the average fitness of the population also shows improvement. The variation operators of mutation and recombination will be discussed in detail in subsequent sections. The new candidates or offspring will compete again based on their fitness and this process is iterated until a candidate with sufficient quality is formed or a previously set computational limit is reached. Table 2.3 shows an iteration of this process, creating new individuals in generation 2.

	Generation 1				Generation 2		
i	**String** X_i	**Fitnes s** $f(X_i)$	$P(X_i)$	**Mating Pool**	**Operation**	**String** X_i	**Fitness** $f(X_i)$
1	010	2	0.14	110	Mutation	111	7
2	101	5	0.36	101	Reproduction	101	5
3	110	6	0.43	110	Crossover	111	7
4	001	1	0.07	001	Crossover	000	0
Total		14					19
Worst		1					0
Average		3.5					4.75
Best		6					7

Table 2.3 One possible outcome of applying genetic operators to create generation 2

This example illustrates how the genetic algorithm created populations with higher average fitness and improved individuals without knowing anything about the problem domain or the fitness measure. In Table 2.2, mutation is seen to have caused a negative effect in fitness whereas in Table 2.3 it shows it to have caused a positive change in fitness. Moreover, the fourth individual has regressed in the second generation and has the worst possible fitness of zero. These are indicative of the random processes that exist in GA. The best individual for this problem is the genotype 111 which corresponds to a crystalline silicon wafer using the epitaxial growth fabrication process to yield the highest profit. Of course, a typical run of a GA may not successfully terminate at its second generation but could possibly run for hundreds or thousands of generations and sometimes not even find the desired solution.

2.2 Representation of Individuals

One of the first steps in the conventional genetic algorithm is to determine the representation scheme. This involves the definition and mapping of genotypes to phenotypes. In nature the chromosomes are character strings in nature's base-4 alphabet, the four nucleotide bases that appear along the length of the DNA molecule adenine (A), cytosine (C), guanine (G) and thymine (T). In GA however, the mathematical objects or genes that undergo evolution are fixed-length character strings. When the values being represented as genes belong to a discrete distribution, the most suitable representation scheme is the integer representation for ordinal attributes such as integers (e.g. {0,1,2,3}) or the cardinal attributes (e.g. {MOSFET, CMOS, JFET, Bi-Polar}). When the genes come from a continuous distribution, the most sensible way to represent the candidate is real-valued strings consisting of *k* genes of floating-point numbers forming a vector $\langle x_1, \ldots, x_k \rangle$ with $x_i \in \Re$. However, the most widely used representation scheme in GA is the binary representation. In this instance, it may be required to decide on how long the bit-string should be or whether the encoding allows a valid representation of all possible solutions. In the previous section, a 3-bit binary string was used for the problem of fabrication of the solid state device. In the following subsections, the binary representation is only considered and discussed.

2.3 Variation Operators

The main aim of the variation operators is to fill the gene pool with "fresh blood". Two of the most common variation operators are discussed here, namely the mutation and recombination operators. Mutation, a unary operator, is the generic name given to operators, which use only one parent to create one child. This involves the application of some kind of randomised unbiased change to the genotype. The most common mutation operator for binary representation involves bit flipping. The parent is chosen probabilistically based on fitness and then the mutation point chosen at random is inverted to create the new offspring. Figure 2.1 illustrates how the specific individual (from Table 2.2) in the problem of fabrication of the solid state device undergoes mutation to create the new offspring.

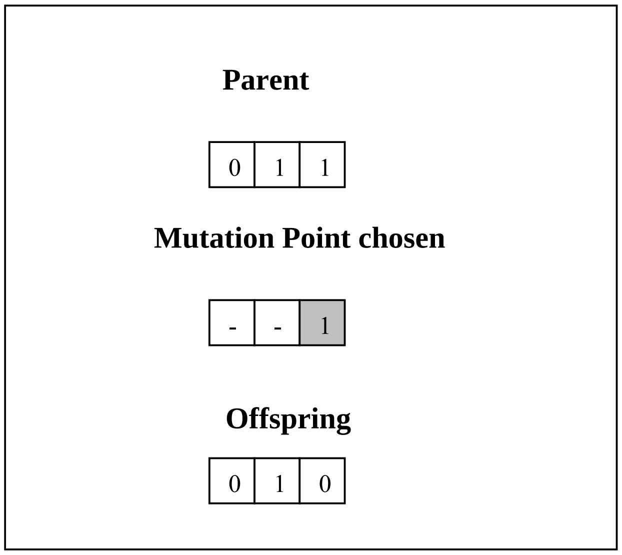


Figure 2.1 Bitwise mutation for binary encoding

From the biological analogy of meiosis, crossover or recombination usually uses two parents to create two children (a binary operator). The most common reproduction operator is the *N*-point crossover. *N* random crossover points are chosen in the range [1,*L*-1], where *L* is the length of the encoding, using a uniform probability distribution. The representation is broken into *N*+1 segments of contiguous genes and alternative segments are taken from two parents to form the offspring. The crossover operation from Table 2.2 in the problem of fabrication of the solid state device is the one-point crossover operation. Two parents are chosen probabilistically based on fitness and then an interstitial point is picked at random splitting both parents at this point and forming two fragments and remainders. As it can be seen in Figure 2.2, the offspring are created by exchanging the tails. In this example, there are *L*-1=2 interstitial locations between the positions of each string and interstitial location 1 is selected as the crossover point.

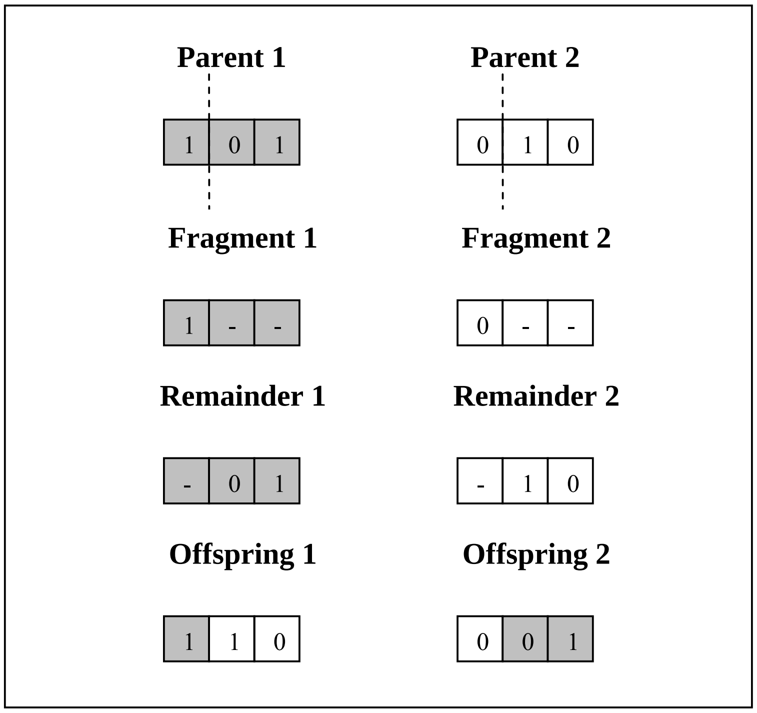


Figure 2.2 One-point crossover for binary encoding

2.4 Some Theoretical Aspects of GA

An exhaustive search strategy is generally non-intelligent and/or non-adaptive. Genetic Algorithm however uses the information, which has been learned about the environment, to control the future direction of the search. It works in a domain-independent way on fixed-length character strings of populations. The total number of points (n_{total}) in the search space is given by equation 2.1, where *K* is the alphabet cardinality and *L* is the length of the character string.

$$n_{total} = K^{L} \qquad (2.1)$$

The total number of possible strings for the problem of fabrication of the solid state device can be represented geometrically as per Figure 2.3, where each corner of the cube represents one of the possible points (2^3=8).

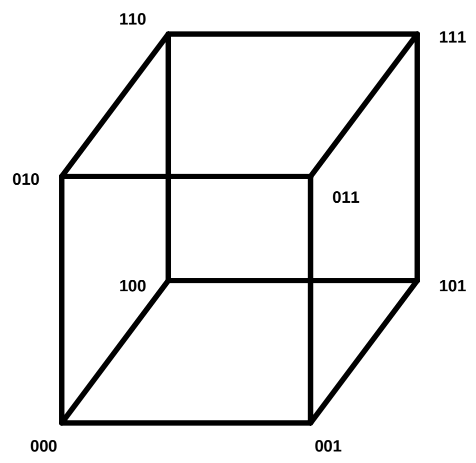


Figure 2.3 Search space for the fabrication of the solid state device problem

A schema H (plural schemata) is a hyper-plane in the search space. Each schema is a string over an extended alphabet of K+1, consisting of the original alphabet and an asterisk the "*don't-care" symbol*. For the example of the binary alphabets the templates can then be represented by the ternary alphabet, namely $\{0,1,*\}$. The schema H is a common representation of these binary alphabets or building blocks, which express a set of points from the search space of a problem, having specific similarities. Some examples of schemata are 101, ***, 1*0 and **1. The Schema Specificity $O(H)$ as defined by (2.2) is the number of positions in the schema that are not defined by a meta-symbol. The number of meta-symbols present within the schema is described by m.

$$O(H) = L - m \tag{2.2}$$

The total number of individuals (NI) that a schemata contains is defined by (2.3) and the number of schemata ($NS_{O(H)}$) of specificity $O(H)$ is defined by (2.4).

$$NI = K^{[L-O(H)]} \tag{2.3}$$

$$NS_{O(H)} = {}^{L}C_{O(H)} K^{O(H)} = \binom{L}{O(H)} \times K^{O(H)} \tag{2.4}$$

The number of crossover points is described by the defining length $\delta(H)$, which is the distance between the outermost specific non-* symbols. For example for the schema H=1**0*1*0, $O(H)$=4 and the defining length $\delta(H)$=7 or δ(*1*0*)=2, δ(1101*)=3, δ(11001)=4 and δ(******)=0. The total number of schemata S_{total} is described by (2.5). For the problem of fabrication of the solid state device, there exist 27 schemata. Figure 2.4, shows 23 of these schemata, where *01 represents 001 and 101.

$$S_{total} = (K+1)^L \tag{2.5}$$

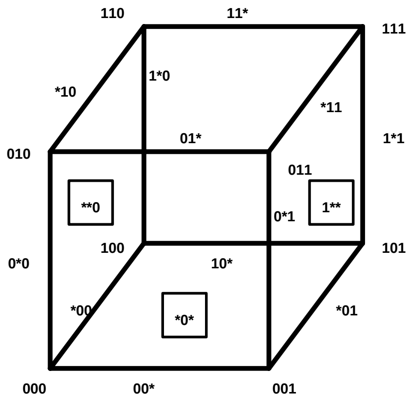


Figure 2.4 Examples of schemata with specificity of one, two and three

2.5 Schema Theorem

Since Holland's pioneering research in standard genetic algorithm using fitness proportionate selection, 1-point crossover and bitwise mutation, the schemata concept has dominated the theoretical analysis of genetic algorithms. The search space can be divided into subspaces, called schemata. Each schema, with its associated fitness, is viewed as a competing explanation. The schemata assemble all the points in the search space that have certain characteristics in common. Schema theorems provide the necessary means to study how and why the individuals in the population move from one subspace to another. Let $f(H,g)$ be the average fitness of a schema H at generation g. For example, $f(H,0)$ denotes the average fitness for the schema *H* at generation 0. Specifically, *f*(*H*,*g*) is calculated as the average of the observed fitness values of the individual strings in the population which belong to the schema *H* as illustrated by (2.6), where *m*(*H*,*g*) is the number of occurrences of schema *H* at generation *g*. It should be noted that *m*(*H*,*g*) is a function of generation *g*. In the initial generation *m*(*H*,0) may be zero but increase in later generation, e.g. *m*(*H*,19) may be nine. A schema theorem provides a mathematical model of why and how *m*(*H*,*g*) varies from one generation to the next.

$$f(H,g) = \frac{\sum_{x_j \in H} f(x_j, g)}{m(H,g)} \tag{2.6}$$

The fitness ratio *FR* of a given schema *H* is calculated as the ratio of the average fitness of a schema *H* to the average fitness $\overline{f(g)}$ of the population at generation *g*.

$$FR(H,g) = \frac{f(H,g)}{\overline{f(g)}} \tag{2.7}$$

The probability of disruption due to crossover ε_c for a genotype of length *L* of a schema *H* is described by (2.8). ε_c is small when $\delta(H)$ is small, indicating that a schema with relatively short defining length generally appears in the future generations with exponentially increasing frequency, as indicated by the schemata theorem (2.10).

$$\varepsilon_c = \frac{\delta(H)}{L-1} \tag{2.8}$$

The probability of disruption due to mutation ε_m for a genotype of length *L* of a schema *H* is defined by (2.9). It should be noted that the simplification can be obtained after the expansion using the Taylor series (Maclaurin series) and approximation by ignoring the high-order terms in p_m. ε_m is small when *O*(*H*) is small.

$$\varepsilon_m = (1 - p_m)^{O(H)} \approx 1 - p_m O(H) \tag{2.9}$$

The schema theorem states that for a GA using the Darwinian operation of fitness-proportionate reproduction and the genetic operations of crossover and mutation, the expected number $m(H,g+1)$ of occurrences of every schema H in the next generation is approximately given by (2.10).

$$m(H,g+1) \geq \frac{f(H,g)}{\overline{f(g)}} m(H,g)(1-\varepsilon_c)(1-\varepsilon_m) \qquad (2.10)$$

A schema would be propagated into subsequent generations at an exponentially increasing rate, when the fitness ratio *FR* over several generations of that particular schema *H* is by at least a steady amount above unity. If ε_c and ε_m are small, the allocation of succeeding trials is approximately optimal. A schema *H* with short defining length and reasonably few defined positions is a building block, which will be proliferated from generation to generation. The genetic algorithm favours the processing of such schemata.

CHAPTER 3

TREE-BASED GENETIC PROGRAMMING

"Give me a lever long enough, and a fulcrum strong enough, and single-handed I can move the world", Archimedes.

3.1 Introduction to LISP

This section provides a brief outline of the LISP (LISt Programming) programming language. The common LISP dialect [293] was initially proposed for the genetic programming paradigm due to its suitability when compared with the other high-level programming languages such as C, JAVA, PASCAL, FORTRAN etc. It was believed that evolving program structures based on JAVA or C++ may not be as straightforward as the LISP programming language. General hierarchical computer programs with varying size and shapes can be expressed using LISP. LISP programs can be easily manipulated as the functions and variables in LISP are defined in the same manner.

There exist two main types of entities in LISP, namely atoms and lists. Atoms comprise of constants and variables. Some examples of constants are the Euler's number (the base of the natural logarithm) e, π or an integer 23. An example of a variable would be the magnetic flux density ***B***, viscosity η or simply the variable *TIME*. A list is an ordered set of items enclosed by parentheses. Typically, a list is a combination of operators and atoms. Two examples of lists are shown in Figure 3.1.

$$\left(+ \ \pi \ \sqrt[3]{2}\right) \qquad\qquad \left(\times \ 6.62\times10^{-34} \ \nu\right)$$

Figure 3.1 Two examples of a list

The common LISP statements are expressed as symbolic expressions (*S*-expression). An *S*-expression can be a list or an atom in LISP. Each S-expression can be classified into functions or operations and terminals or arguments. Each function will require argument(s), and has the structure of a list. Each terminal is a single atom (variable or constant atom) that can be evaluated immediately. The execution of the LISP S-expression follows the Polish notation, commonly known as prefix notation. If the arguments in the S-expressions are lists rather than atoms, the execution is in a recursive depth-first manner starting from left in the nested parentheses. Figure 3.2 shows a complex composite function in LISP illustrating this idea.

$$\begin{gathered}
(+(\times\sin(x)\sin(x))\ (\times\cos(x)(\frac{\partial}{\partial x}(\sin(x))))\\
\Downarrow\\
(+(\times\sin(x)\sin(x))\ (\times\cos(x)\cos(x)))\\
\Downarrow\\
(+(\sin^{2}(x)\ \cos^{2}(x))\\
\Downarrow\\
1
\end{gathered}$$

Figure 3.2 An example of execution of composite functions in LISP

Any LISP expression can be graphically presented as a rooted point-labelled tree with ordered branches, commonly known as a parse tree. The S-expression in Figure 3.2 is shown as a parse tree in Figure 3.3. As it can be seen from Figure 3.3, there are four leaf nodes (external points) labelled with terminal (variable atom) *x* and eight internal points labelled with functions +, ×, sin, cos and ∂/∂x. Unless the parse tree is a single atom, the root node of all parse trees is always an operation or function. All the leaf nodes are always an atom. Each node within the parse tree can have as many children nodes as required. In this example, some of the functions have one or two children and the terminals have obviously no children.

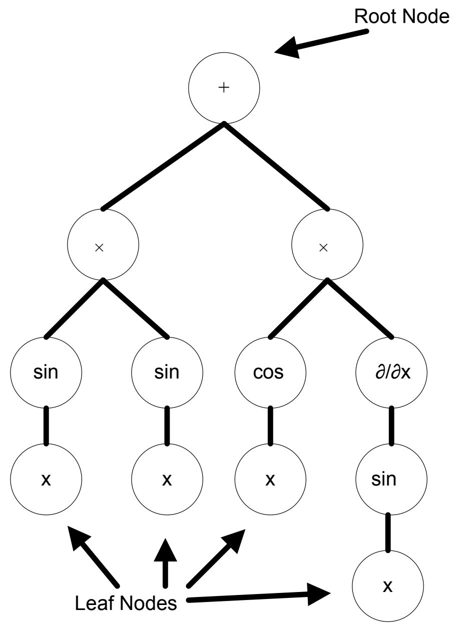


Figure 3.3 Parse tree structure of a composite function

3.2 Structures Undergoing Adaptation

As discussed in the previous chapter, Genetic Algorithms (GA) typically apply biologically inspired evolutionary operators to fixed length binary characters. Genetic programming (GP) extends this by increasing the complexity of the structures that undergo adaptation to broad hierarchical computer programs with dynamically varying forms and sizes. In the classical GP, the structures are typically comprised of the set of N_{func} functions from the function set $F = \{f_1, f_2, \cdots, f_{Nfunc}\}$ and the set of N_{term} terminals from the terminal set $T = \{a_1, a_2, \cdots, a_{Nterm}\}$ forming the combined set $C = F \cup T$.

Definition 3.1 The **terminal set** consists of all the inputs and the constants supplied to the GP algorithm together with the zero argument functions with side-effects executed by GP.

Definition 3.2 The **function set** consists of all the operators, statements and functions supplied to the GP algorithm.

Definition 3.3 The **combined set** is the union of the terminal set and the function set, i.e. $C = F \cup T$.

The functions and terminals are specific for the problem at hand. The functions in the function set may include mathematical functions, conditional operators, iterations, recursions or any domain specific function and the terminals are variable atoms or constant atoms. For example, for a sequential logic problem of designing a JK flip-flop, the combined set may be $C = F \cup T = \{AND, NAND, NOR, OR, NOT, J, K, Q\}$, whereas the problem of finding the solution to the transverse magnetic waves in rectangular waveguides may have a combined set of $C = F \cup T = \{\cosh, \sinh, \sin, \cos, \ln, x, y, a, b, \pi, m, n\}$. Structures are typically represented as a rooted, point-labelled tree with ordered branches. For example, the solution to the JK flip-flop problem produced by the GP process is shown in Figure 3.4 and expressed in disjunctive normal form (DNF) by the following LISP S-expression (*OR* (*AND J* (*NOT* (*NOT* (*NOT* (*Q*))))) (*AND* (*NOT* (*K*)) (*NOT* (*NOT* (*Q*)))))).

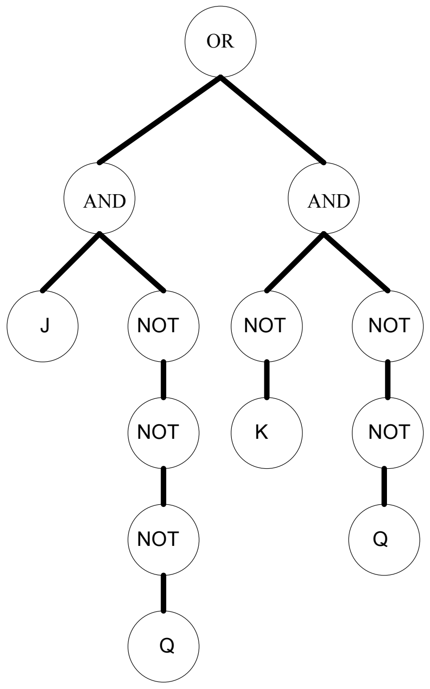


Figure 3.4 Parse tree of JK flip-flop as rooted point-labelled tree with ordered branches

The space of all the possible LISP S-expressions composed of elements from the combined set for the specific problem forms the search space for GP. The nine internal points of the tree are labelled with functions *OR*, *AND*, *AND*, *NOT*, *NOT*, *NOT*, *NOT*, *NOT* and *NOT*. The four external points or leaves of the tree are labelled with terminals, the Boolean variable atoms *J*, *K*, *Q* and *Q*.

Each node in the parse tree can contain as many children nodes as required contributing to different parse tree shapes. Moreover, the shape and size of a generated parse tree is also dependent on the depth of the tree. Due to computer memory limitations, a size restriction or limit is typically placed on the depth of the tree. The depth of a parse is however constrained by (3.1) [46], assuming that $n \geq 1$ and $t \geq 2$:

$$depth \leq \log_t \left(\frac{n+1}{2} \right) \qquad (3.1)$$

The minimum degree t refers to the minimum bound on the number of children contained within each node and n represents the number of children nodes linked to a single parent node.

3.2.1 Closure Property

It is desired that the terminal set and the function set in the genetic programming satisfy the *closure property*. The closure property requires that all the functions $f \in F$ can accept as their arguments any terminals $a \in T$ and any data type returned by any function $f \in F$. For example, for the arithmetic operation such as division by zero or mathematical function such as logarithm of non-positive numbers, the closure property is not satisfied and hence a protected division operator *%* or protected logarithm *PLOG* will need to be defined. If the closure property is not met, the individuals will need to be discarded if they do not evaluate to an acceptable result.

3.2.2 Sufficiency Property

It is incumbent that some composition of terminals $a \in T$ and functions $f \in F$ will yield a solution to the problem. This is known as the sufficiency property, where it is required to identify functions and terminals with sufficient power to solve a particular problem. Determining the repertoire of primitive functions and terminals is considered one of the most important preparatory steps in GP, but it is common to virtually every problem in science and other machine learning paradigms.

3.3 Initialisation

The initial population is created by randomly generating individual S-expression of rooted, point-labelled trees with ordered branches. Using a uniform random probability distribution, the selection of the root of the tree is restricted to a function $f \in F$, as it is desired to produce hierarchical initial structures rather than a degenerate structure comprised of a single terminal. For every point within the tree with a function $f \in F$, there will be $a(f_i)$ lines radiating out from the respective node, where $a(f_i)$ returns the arity of f_i (or number of arguments f_i takes). For each of these radiated lines an element is randomly selected from $C = F \cup T$ using a uniform random probability distribution to be the endpoint of that radiating line. If a function is chosen, then the above steps are recursively iterated. However, if a terminal is selected for that point, that point becomes the endpoint for the tree and the generating process is consequently terminated. There exist three main generative process implementation methods which are discussed in the following subsections.

3.3.1 FULL Method

The *FULL* method restricts the selection of nodes n at depths less than the maximum depth ($d(n) < D_{\max}$) to a function $f \in F$ and nodes at the maximum depth to a terminal $a \in T$. The depth of a node $d(n)$ is the length or number of branches connecting the specific node to the root node and the depth of a tree is defined as the length of the longest non-backtracking node from the root to the endpoint. The maximum depth of any rooted point-labelled tree with ordered branches is denoted by $D_{\max}$. The FULL method can be summarised as per (3.2).

$$n = \begin{cases} f \in F, \forall n \ s.t. \quad d(n) < D_{\max} \\ a \in T, \forall n \ s.t. \quad d(n) = D_{\max} \end{cases} \tag{3.2}$$

The *FULL* method produces a full parse tree, where the tree is fully balanced and the left and right hand side of the root node has the same amount of nodes and the same depth.

3.3.2 GROW Method

As opposed to the *FULL* method, the *GROW* method generates random trees that are variably shaped. Each node *n* at depths less than the maximum depth ($d(n) < D_{\max}$) is randomly selected from the combined set $C = F \cup T$, whereas nodes at maximum depth ($d(n) = D_{\max}$) are restricted to a terminal $a \in T$. The trees produced are not balanced and are variably different in shape and sizes.

3.3.3 Ramped Half-and-Half Method

The *Ramped Half-and-Half* method is the most popular method in GP and is a mixture of the previous two methods. It incorporates both the *GROW* method and *FULL* method by generating equal numbers of trees from each of the methods and thereby maximising the variety of trees in the population. The depth of the tree ranging from 2 to $D_{\max}$ is used as a parameter to create trees. To illustrate, if the initial maximum tree size is limited to 9, then 12.5% of the population will have a depth of two, 12.5% will have a depth of three and so on. For each value of depth 50% of the trees are generated using the *FULL* Method and 50% using the *GROW* method. This means that for example 6.25% of trees have a depth of three and are generated using the *FULL* Method. There is a chance that random trees may be identical, wasting computational resources and undesirably reducing genetic diversity. It is desirable to remove these duplicates in the initial random population but it is not necessary. It should be noted that in the classical genetic programming checking for uniqueness is not performed.

3.4 Fitness

In nature, the driving force of Darwinian natural selection is fitness. Similarly, in the genetic programming paradigm fitness measures the performance of an individual with respect to solving a particular problem and thereby determines the fate of that individual and whether the individual will survive and propagate into the next generation. It can be measured implicitly or explicitly. Most commonly, fitness is measured by creating an explicit fitness measure for each individual in the population. In other words, each parse tree is assigned a scalar value by using a well defined explicit evaluative procedure.

3.4.1 Raw Fitness

Raw fitness is a measure of performance in the natural terminology of the problem itself. For example, in the artificial ant problem the raw fitness is the number of food pellets eaten by the ant. The more pellets the ant eats the better. For many problems, a single fitness value does not accurately describe the performance of an individual. In these instances, more fitness cases are needed for the generalisation of the problem domain. For example, for the cart centring problem, twenty different initial conditions of position and velocity are chosen for the problem [170]. The fitness is then the sum of time over the twenty fitness cases to centre the cart. The shorter the time, the better the algorithm is to centre the cart. The most common definition of raw fitness $r(i,g)$ of an individual i at generation g is given by (3.3).

$$r(i,g) = \sum_{j=1}^{N_{fc}} \left| S(i,g) - C(j) \right| \tag{3.3}$$

$S(i,g)$ is the value returned by the S-expression, $C(j)$ is the correct value for the fitness case j and N_{fc} is the total number of fitness cases.

3.4.2 Standardised Fitness

As it could be seen from the examples of the previous subsection, the best value of raw fitness may be large or small depending on the problem, as raw fitness is stated in the natural terminology of the problem. The standardised fitness $s(i,g)$ expresses raw fitness $r(i,g)$ in such a way that a lower numerical value is always a better value. The standardised fitness is governed by (3.4), where $r_{\max}$ is the maximum possible value of raw fitness for a given problem.

$$\begin{cases} s(i,g) = r(i,g) & \text{, for minimisation problems} \\ s(i,g) = r_{\max} - r(i,g) & \text{, for maximisation problems} \end{cases} \tag{3.4}$$

3.4.3 Adjusted Fitness

The adjusted fitness translates the possible lower and upper limits of the fitness value to 0 and 1. The higher the adjusted fitness $a(i,g)$, the better the individual in the population. The adjusted fitness is computed as follows.

$$a(i,g) = \frac{1}{1 + s(i,g)} \tag{3.5}$$

The main benefit of this fitness measure is that it exaggerates the importance of small differences between standardised fitness values, as they approach the value zero. This way a greater emphasis is placed between a good individual and a very good one.

3.4.4 Normalised Fitness

The normalised fitness $n(i,g)$ is computed as per (3.6), where M is the population size.

$$n(i,g) = \frac{a(i,g)}{\sum_{k=1}^{M} a(k,g)} \tag{3.6}$$

As the name implies, this fitness measure is normalised and therefore the sum of all the normalised fitness values is one and its range is between 0 and 1. In addition, the higher the $n(i,g)$, the better the individual in the population.

3.5 Selection Schemes

The selection scheme can play a significant role within GP and may ultimately determine the diversity of the population. All selection schemes have a common objective, where they mimic Darwinian natural selection and pick individuals based on their fitness value.

3.5.1 Parent Selection versus Survivor Selection

Parent selection [170] is a probabilistic process where individuals with higher quality are selected to become parents of the next generation. Selected individuals undergo variation to produce offspring. Survivor selection [170] is the process where individuals with higher fitness are favoured to participate in the next generation replacing an old number of individuals from the previous generation. The survivor selection and parent selection mechanisms are responsible for enforcing quality improvements.

3.5.2 Fitness Proportionate Selection

Fitness proportionate selection (FPS) introduced in [119] selects individuals $X_g(i)$ at generation g based on their fitness $f(X_g)$ according to the probability $P(X_i)$ defined by (3.7).

$$P(X_i) = \frac{f(X_g(i))}{\sum_{j=1}^{M} f(X_g(j))} \tag{3.7}$$

Although one of the merits of this selection scheme is its simplicity and it has widely been used in the traditional GP, there are some shortcomings associated with FPS. Individuals with highest fitness can be heavily selected leading to premature convergence. In addition, when the fitness values of the individuals are closely clustered, the selection pressure virtually vanishes leading to almost uniform random selection.

3.5.3 Ranking Selection

The ranking selection addresses the drawbacks of FPS by maintaining a constant selection pressure. Individuals are sorted on the basis of fitness and selection probabilities, which are allocated to them according to their rank. There are various probability allocation schemes such as linear and exponential. In the Linear Rank selection, the rank of individuals ranges from 1 to *M*, i.e. [1,*M*], where the best individual receives a rank of *M* and the worst individual a rank of 1. After the allocation of the ranks, the selection processes follows FPS as per (3.7). The amount of selection pressure is limited for linear mapping. The Exponential Ranking selection provides the means to place more emphasis on the selection of individuals with higher fitness values. The probability of selection for the exponential ranking scheme is governed by (3.8), where σ is the normalisation factor ensuring that $\sum_{i=1}^{M} P(X_i) = 1$.

$$P_{\exp-rank}(X_i) = \frac{1 - e^{-i}}{\sigma} \tag{3.8}$$

The main advantage of rank selection is that it exploits the small differences between individuals, maintaining population diversity and avoiding premature convergence.

3.5.4 Tournament Selection

For the ranking selection and FPS, the fitness values of entire population is required. In some instances obtaining this knowledge becomes a formidable task, where it becomes highly time consuming to determine the fitness of all individuals in the population. This can be seen when the population is distributed on a parallel system or when the population size is very large. There are instances where the determining universal fitness is almost impossible such as in evolving applications for game playing. Tournament selection can surmount these difficulties as it does not require a global knowledge of the population. The tournament selection selects k number of individuals from the population M and the best amongst the k individuals is selected for propagation (deterministic tournaments). To increase the chances of selection of members with the above-average fitness, the tournament size k should be increased. Binary tournaments (k=2) are most broadly used in GP.

3.5.5 Greedy Over-selection

In the greedy over-selection, individuals with higher fitness are greedily overselected as opposed to the standard FPS. It is believed that complex problems require a larger population size M [170]. Greedy over-selection is typically used for problems with large M. This scheme is implemented by ranking the individuals based on fitness and then dividing them into two groups, the fittest individuals (group I) containing the top b% and the less fit individuals (group II) containing (100- b)% of the individuals. 80% of the selection operation are from group I in proportion to normalised fitness and 20% from group II . The values of b depend on population size M and are determined empirically. The “rule of thumb” values of b are documented in [71] and shown in Table 3.1 below.

M	***b***
1000	32%
2000	16%
4000	8%
8000	4%

Table 3.1 Proportion of subpopulation in the fitter group

3.5.6 Age-based Replacement

Age-based replacement [71], mainly a surviver selection scheme, takes into account the age of the population rather than its fitness. This strategy implements a first-in-first-out (FIFO) queue, where individuals within a certain threshold (age) are the only candidates suitable for selection and individuals beyond this threshold are simply discarded.

3.5.7 Elitism

Elitism [170] prevents the loss of the fittest individual within the population by ensuring that the current fittest individual always survives and is kept in the population. Thus the fitness of the best individual is an increasing function.

3.5.8 Genitor

The genitor scheme [170] replaces the worst *k* members of the population with new individuals. Although the mean population fitness can rapidly increase when this scheme is chosen, it can at the same time lead to premature convergence.

3.6 Evolutionary Operators

The evolutionary operators in GP are reproduction and the variation operators. The role of the variation operators is to produce new offspring from the current individuals. The main evolutionary operators in GP are discussed in this section.

3.6.1 Reproduction

The reproduction operator, a primary operation, is an asexual operator which takes one parental *S*-expression and creates only one offspring *S*-expression. This operation is conducted in two steps in which an individual is selected and then copied without any alteration into the new population. The number of individuals subjected to this operation is controlled by the probability of reproduction P_r.

3.6.2 Crossover

Another primary operation is the sexual recombination or crossover, where it produces new offspring by swapping genetic material between the selected parents, as depicted in Figure 3.5. Crossover is the predominant search operator in genetic programming. The crossover operation is highly stochastic. Crossover is a binary operator, where it selects two *S*-expressions probabilistically and produces two new *S*-expressions. One point in each parent is randomly selected to become the crossover points. In selecting the crossover point, a higher probability P_{ip} is given to the internal (function) points of the tree. This distribution promotes the recombination of much larger structures. The crossover fragment of each parent is itself a rooted subtree, with its root being the crossover point. The offspring is created in a symmetric manner, where the crossover fragment of the first parent is deleted and the crossover point of the other parent is inserted at the crossover point of the first parent. As it is required for all functions to comply with the closure property and entire subtrees are exchanged, this genetic operation creates syntactically legal LISP *S*-expressions. The maximum depth of a tree during the evolution $D_{\text{evolution}}$ limits the maximum permissible size of a tree. If the crossover operation results in offspring of impermissible size, the crossover operation is aborted and one of its parents is arbitrarily selected to be reproduced.

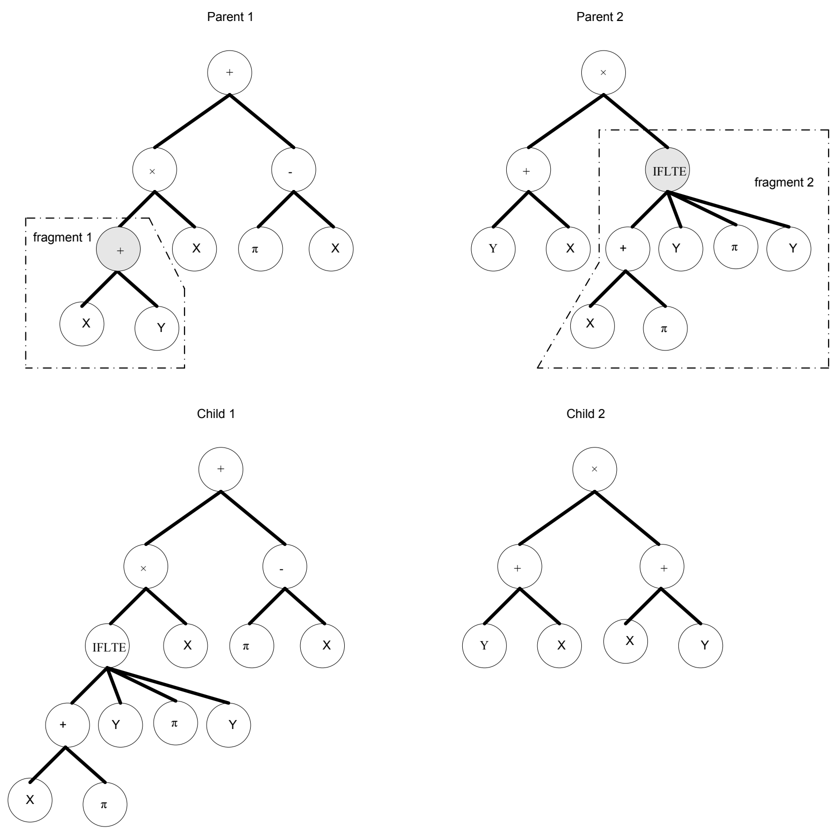


Figure 3.5 Illustration of recombination - Parental programs, crossover fragments and the resulting offspring

3.6.3 Mutation

Mutation, a unary operator, introduces random changes in the individual. In the conventional genetic algorithm, there is a possibility that an allele (particular symbol) at a specific position on a chromosome string to disappear and become extinct. In such algorithms the mutation operation plays a significant role and is therefore the primary operator. In genetic programming however, mutation is considered as a secondary operator, as it is particularly rare for an element to disappear in GP. Moreover, the crossover operation becomes a point mutation, when the crossover points of the two parents are endpoints of their respective trees. Mutation, an asexual operator, is implemented by selecting one parental *S*-expression probabilistically and then randomly selecting a node within this tree. The mutation point can be an internal or an external point. The mutation operation removes the selected mutation point as well as the entire sub-tree lying below the mutation point and then appends a randomly generated sub-tree at that point. Note that the depth of the child tree can exceed that of its parent. The maximum permissible tree size is limited by $D_{\text{evolution}}$. Figure 3.6 illustrates the mutation operation.

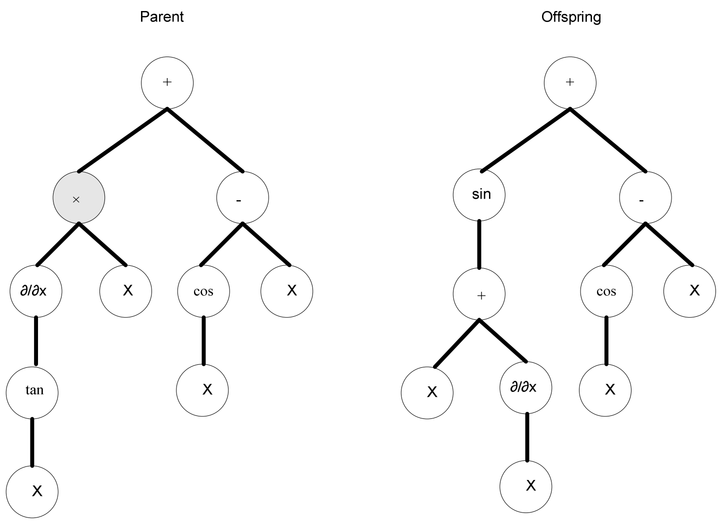


Figure 3.6 Illustration of mutation

3.6.4 Permutation

In genetic algorithms, the inversion operation reorders alleles exploring the genetic linkage between combinations of alleles, with the hope of bringing certain alleles closer together to increase performance. Permutation, an asexual operation, selects an individual randomly and then selects an internal point within the LISP *S*-expression at random. For a function with *k* arguments, a permutation is randomly selected from the set of *k* factorial (*k*!) possible permutations. Permutation has no immediate effect, if the function is commutative. Permutation is considered a secondary operation and is not generally used in GP. Figure 3.7 illustrates an example of permutation.

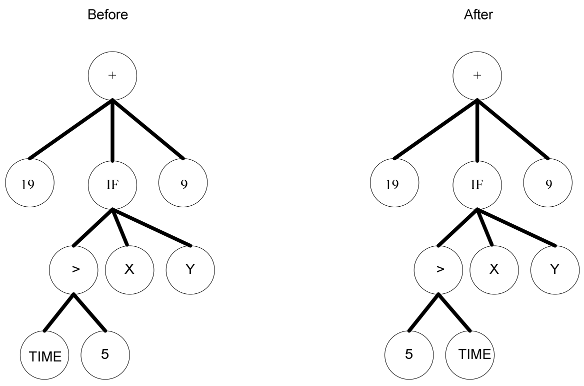


Figure 3.7 Illustration of permutation

3.6.5 Editing

Editing, an asexual operation, selects one parental *S*-expression and produces one more parsimonious offspring *S*-expression. The editing operation uses a selected set of domain independent and domain specific editing rules to simplify the S-expression. Figure 3.8 shows examples of editing in Boolean domain. The last example shows the application of De Morgan's law to the symbolic S-expression.

(AND X X)	→	X
(OR X X)	→	X
(AND (NOT X) (NOT Y))	→	(NOT (OR X Y))

Figure 3.8 Editing – Examples in Boolean Domain

A frequency parameter f_{ed} controls the amount of the editing operation, where the editing operation is applied to all generations when $f_{\mathrm{ed}}=1$ and to no generations when $f_{\mathrm{ed}}=0$. The editing operation is sometimes considered to be very time consuming and a secondary operation. Editing is not extensively used in GP, as the question of whether breeding for parsimony will degrade performance due to prematurely reducing variety or will be beneficial still remains unclear.

3.6.6 Encapsulation

Encapsulation, an asexual operation, selects one parental *S*-expression and produces one offspring *S*-expression. After selecting an individual tree randomly, it selects an internal point at random and removes the sub-tree underneath this selected point. It then defines a new encapsulated function, which has no arguments and is named E0, E1, E2, etc. and allows references to the deleted sub-tree. The function set is augmented to include these newly defined functions. The newly defined functions can now for example be used during the mutation operation. The motivation behind this operation is that potential building blocks can be created for future generations that cannot be subjected to the disruptive effects of crossover, as this new building block is now considered to be an indivisible single atom. This operation is not generally used in GP.

3.7 Control Parameters

The parameters that control the execution of the classic genetic programming paradigm are discussed in this section. Two of the major numerical parameters are the population size *M* and the maximum number of generations *G* in one run. There exist nine minor numerical parameters controlling the GP process, namely the probability of crossover P_{c}, the probability of reproduction P_{r}, the probability of mutation P_{m}, the probability of permutation P_{p}, the probability of selecting an internal point as a crossover point P_{ip}, the maximum initial tree depth for the initial population D_{initial}, the maximum depth of a tree during the evolution $D_{\text{evolution}}$, the frequency of editing f_{ed}, the probability of encapsulation P_{en}. Some of the qualitative parameters that affect GP are the generative method for the initial random population and the different selection methods used.

3.8 Preparatory Steps

Computer programs are entities that perform computations on received inputs to produce some form of outputs (Figure 3.9). The computations could involve basic arithmetic, conditional operations, iterations, recursions, storage of information in memory, building reusable groups into subroutines or passing and receiving information to/from subroutines.

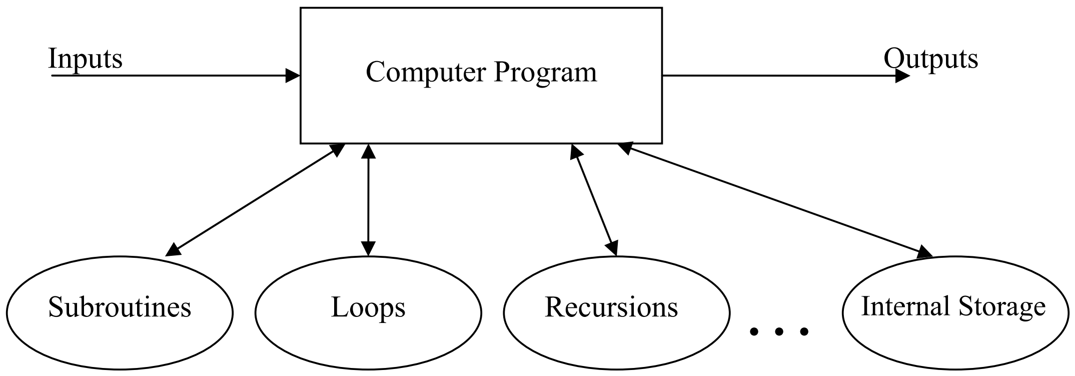


Figure 3.9 A Computer program

A system that automatically creates computer programs must possess a number of attributes and capabilities. For the production of automatic computer programs it is necessary to start off with a high-level statement citing all the requirements. The system is required to produce an entity that can be executed on a computer producing results which satisfactorily solve the problem. There exist five major preparatory steps in the standard genetic programming (SGP). These preparatory steps form the communication of the high-level statement of the problem to the GP system and are outlined below and summarised in Figure 3.10.

The five major preparatory steps are:

i. Determine the set of terminals that are needed to solve the given problem.

ii. Identify the set of primitive functions.

iii. Formulate and establish the fitness measure.

iv. Set the values for the control parameters.

v. Define the termination criterion and the method for the result designation for the run.

The first two steps determine the ingredients that will make up the space of all computer programs that may solve or approximately solve the given problem. The third step is the chief mechanism for conveying a high-level statement of the problem requirements. The last two steps are administrative.

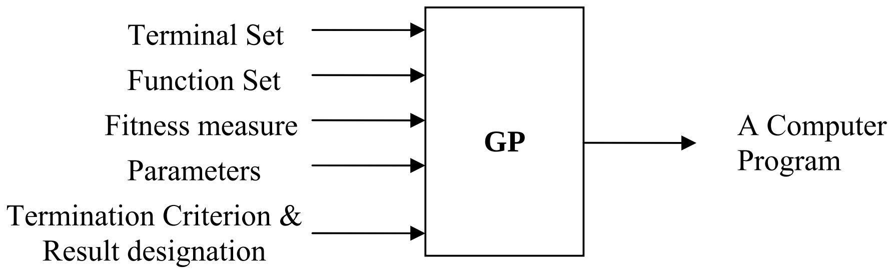


Figure 3.10 Five preparatory steps for SGP

GP is reminiscent of nature in which it is a never ending process. For practical reasons, it is therefore required to define possible criteria that when satisfied will result in termination of GP. Typically, the generational predicate or some problem specific success predicate is required to be met before the run is terminated. In general, the best individual (the *best-so-far* individual) is designated as the result of the GP process.

CHAPTER 4

AN INTRODUCTION TO THE THEORY OF GENETIC PROGRAMMING

"One may say the eternal mystery of the world is its comprehensibility", Albert Einstein.

4.1 Some brief notes

In this chapter, some of the theoretical basics in the field of genetic programming are reviewed. In addition, some of the challenges that the GP paradigm faces are briefly discussed. This chapter provides the necessary foundations for the next chapter, which discusses the innovations that have been proposed to remedy the problems that GP encounters and to improve GP.

4.1.1 The Turing Machine

The Turing Machine TM [308] is a simple, inefficient computer that can simulate the behaviour of any other computer using a finite set of states. A set of state transition rules define specific actions that the TM should perform depending on the input it reads and its current state. It should be noted that it is different to a Von Neumann Machine, which is a computer where the program and the data used by that program reside in the same storage. Any programming language that can emulate the behaviour of the TM is said to be Turing complete.

4.1.2 Convergence

Convergence can be generally interpreted as the point at which the population contains a considerable number of similar individuals. In this instance, the algorithm is either not progressing satisfactorily or is approaching a local optimum (premature convergence). At times, convergence has been interpreted as the point at which the algorithm is approaching the global or optimal solution. In the former, convergence could be considered to be a serious problem or weakness as it could be expected that after repeated cycles of the evolutionary process uniformity may arise sooner or later. Maintaining diversity may be considered a possible remedy to this problem.

4.1.3 No Free Lunch Theorem (NFL)

The No Free Lunch Theorem (NFL) [317] states that no search algorithm is superior to any other algorithm on average across all possible problems. For example, GA may perform better than random search for certain problems, whereas random search may outperform GA for other different problems. The erroneous conclusion should not be made from this that there is no point in designing better algorithms or improving algorithms. We are not typically interested in solving all possible problems but rather a certain class of problems that are suitable for the evolutionary algorithms to tackle. The broad implication of the NFL in regards to performance improvement of the algorithms is that the modifications may only be appropriate for certain form of representations or the modifications may provide improved performance for a specific class of problems. This would mean that the stated improvements are always accompanied with certain assumptions and limitations.

4.2 The Crossover Operator

Any tree or sub-tree within the population can be nominated as a *building block*. Individuals that contain good building blocks have improved fitness values. Therefore, these individuals have a higher probability of selection. Consequently, good building blocks are likely to increase and spread as they are swapped among individuals. The controversy about the effectiveness of crossover relates to whether crossover disrupts or preserves good schemata. It should be noted that most of the theoretical analysis of the crossover operator depend on this balance between disruption and preservation of schemata.

4.2.1 A Gedanken Experiment

If the dark nodes (nodes 10-12) in the Figure 4.1 represent a good building block and the crossover is randomly distributed across the 18 nodes (excluding the root node), the probability that the good building block is disrupted is $2/18 \approx 11\%$. However, if the nodes 6 to 14 represent a good building block this probability is increased to approximately 44%. Now if it is assumed that all the highlighted nodes represent a good building block, the probability that the good building block is disrupted is $13/18 \approx 72\%$.

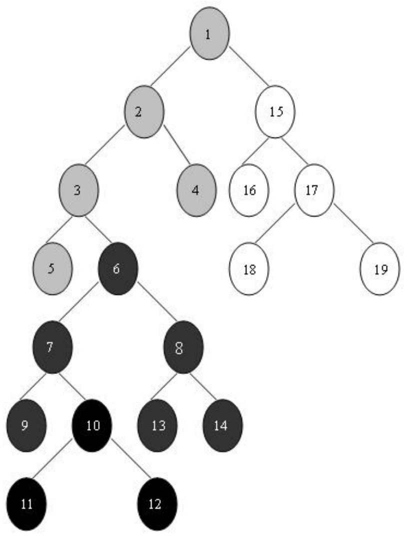


Figure 4.1 Demonstrating the destructive effects of crossover

As it can be seen from the above experiment, as the building block grows larger and larger it becomes more fragile, because it becomes more prone to being broken by the GP crossover operator. In fact, if it is assumed that the building block in the above figure is almost perfect and all that is needed is to discard the white nodes, which is e.g. replaced with a terminal node by the crossover operator, now under this scenario the probability of disruption becomes 13/14 ≈ 93%. It should also be noted that a good building block which has not been disrupted by crossover would still need to be inserted into an individual where its effect is still upheld. This means that the insertion would be context dependant. It has been shown in the literature that crossover has a crushing negative effect on the fitness of the children relative to their parents. It should be mentioned that this is in strong contrast to the biological crossover. In addition, it is believed that the destructive effect of crossover is the main culprit as the cause of the bloat effect in GP.

4.3 GP Introns

An intron is a feature of the genotype that does not affect the fitness of the individual. The theoretical evidence suggests that introns emerge in response to the destructive effects of genetic operators. It should be noted that it is not argued that destructive operators are always bad or that it is always necessary to produce children that have higher fitness than their parents. Such a suggestion would reduce the GP paradigm to a simple hill climbing.

At the end of the run, individuals are close to their best performance. At this time it is very hard for individuals to improve their fitness. Consequently, their strategy for survival will change. They attempt to protect themselves from the destructive variation operators by introducing introns within them at an exponential rate, leading to the phenomenon of bloat. This explosive growth of introns leads to neutral crossover. At this point, the individuals are simply exchanging introns.

4.3.1 Effective Fitness

Definition 4.1	The **absolute length** or **absolute complexity** of a program is the total size of the program.
Definition 4.2	The **effective length** or **effective complexity** of a program is the length of the active parts of the code within the program that affects the fitness of the individual, which excludes any introns.
Definition 4.3	The **probability of destructive crossover** is the probability that a crossover in an active part of a program leads to a worse fitness for the individual. By definition the probability of destructive crossover of an absolute intron is zero.

The effective fitness is related to the chance that variation operators will affect the fitness of the parent's offspring. The average proportion $P_\text{i}(g+1)$ of any individual i in the next generation when using fitness proportionate selection is given by:

$$P_i(g+1) \approx P_i(g)\frac{f_i}{\bar{f}(g)}(1 - p_c \frac{C_i^e}{C_i^a} p_i^d) \qquad (4.1)$$

, where

C_i^e is the effective complexity of program i and C_i^a its absolute complexity,

p_c is the probability of crossover and f_i is the fitness of the individual i,

$\bar{f}(g)$ is the average fitness of the population at generation g,

p_i^d is the probability of destructive crossover.

This equation implies that the proportion of a program in the next generation equates to the proportion of the same individual under the selection operator minus the proportion of these programs disrupted by the crossover operator. It should be noted that this is a conservative measure as it does not cater for the scenario where the individual *i* may be recreated as a result of the crossover operator. The effective fitness of a parent tree measures how many offspring of that parent can be selected for reproduction in generation *g*+1 and is given by:

$$f_i^e = f_i - f_i p_c \frac{C_i^e}{C_i^a} p_i^d \tag{4.2}$$

The effective fitness can increase by either reducing the effective complexity or by increasing the absolute complexity. Whilst there are arguments that introns are beneficial [22] [28] [242] [243], at the same time there are arguments that introns lead to poor performance [6] [109] [115] [209] [303]. It should be noted that this field is very young and more study is required before any conclusive statements can be made. It is believed that the phenomenon of bloat (the exponential growth of introns) is not desirable, as no significant improvements can be made and the effective growth has ended, resulting in the stagnation of the run. Although introns produce large programs, they may encourage parsimony in the effective solution. This observation can be made from (4.2), as when C_i^e and C_i^a are equal (no introns) the influence of destructive crossover is at its maximum but when there are introns any reduction in C_i^e increases the effective fitness. In addition, introns may protect useful genes against crossover in the early stages of the evolution, attracting crossover and facilitating the exchange of good functional blocks.

4.4 GP Schemata and Schema Theorems

Schemata and schema theorems are frequently used to describe why genetic algorithms work. A schema theorem explains how schemata, which are similarity templates representing groups of chromosomes, propagate from one generation to the next under the effects of evolutionary processes. The definition of a schema for GP is much less straightforward than for GAs. The complexity of transferring the schema theorem from GA to GP is due to its variable length representation and the movement of genetic material in the genome from one location to another in GP. Several alternative definitions of GP schemata have been proposed in the literature. A schema is defined as a similarity template composed of one or multiple trees or fragments of trees. It should be noted that none of the existing formulations of a GP schema theorem can predict with any certainty that good schemata will propagate during a GP run. In the next three subsections, schemata are interpreted as components within the program tree which can propagate through the population. They model how the number of instances of such components varies over every generation. In the remaining subsections, schemata are interpreted as subsets of the search space and they model how the number of individuals in such subsets varies over every generation. The first five schema theorems, discussed in the following subsections, give only a lower bound for the expected number of individuals belonging to a schema at generation g+1.

4.4.1 Koza's GP Schemata

The first attempt to use Holland's schema theorem in GP was made in [170]. A schema H is represented as a set of S-expressions, defining a subspace of all trees. For instance, the schema $H = [\ (\ \%\ 2\ X\ )\ ,\ X\ ]$ represents all programs that include at least once occurrence of X and at least once occurrence of $(\ \%\ 2\ X\ )$.This does not cater for the position of the schema, rather its defining components. Consequently, the same schema can be instantiated in various ways in the same program. An example of a program $(\ \times\ X\ \ (\ \%\ 2\ X\ )\ )$ which matches the above schema is shown in Figure 4.2. The arguments in [170] were informal and did not provide any ordering or length definition for this schema.

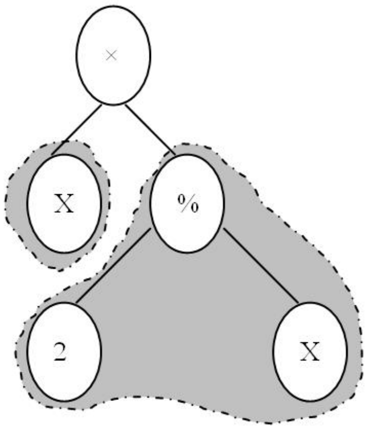


Figure 4.2 An example of Koza's schemata

4.4.2 Altenberg's GP Schema Theorem

The first mathematical formulation of a schema theorem for GP was performed in [2]. It was assumed that the population was very large and fitness proportionate selection was used but no mutation. Thus, the frequency of a program *i* in the next generation (*g*+1) is given by:

$$\begin{aligned}\frac{m(i,g+1)}{M} &= (1-p_c)\frac{f_i}{\bar{f}(g)}\frac{m(i,g)}{M} + \\ & p_c \sum_{j,k\in\Phi}\frac{f_j f_k}{\bar{f}^2(g)}\frac{m(j,g)}{M}\frac{m(k,g)}{M}\sum_{s\in\Omega}P(i\leftarrow j,s)C(s\leftarrow k)\end{aligned} \tag{4.3}$$

, where

Φ is the space of the population,

Ω is the space of all possible sub-expressions extractable from Φ,

$P(i \leftarrow j, s)$ is the probability of inserting expression *s* in program *j* to create program *i*,

$C(s \leftarrow k)$ is the probability that crossover picks expression *s* (schema) in program *k*.

Equation (4.3) models the propagation of programs under the standard crossover operator and assumes that one offspring is only produced as a result of crossover. It is only valid for the infinite population limit but can be generalised as per (4.4) and be valid for both finite and infinite populations.

$$E\left[\frac{m(i,g+1)}{M}\right] = (1-p_c)p(i,g) + p_c \sum_{j.k\in\Phi} p(j,g)p(k,g) \sum_{s\in\Omega} P(i \leftarrow j,s)C(s \leftarrow k) \qquad (4.4)$$

,where

$p\ (x\ ,\ g)$ is the probability of selection of program x at generation g,

E [ ] is the expected value operator.

A schema s in this definition is a sub-expression. It differs from the Koza's schema, where it can be made from multiple sub-expressions.

4.4.3 O'Reilly's GP Schemata

A schema is defined [247] as a multiset of sub-trees and tree fragments, which are trees with at least one leaf that is a "*don't care*" ("*") symbol which can be matched by any sub-tree. For instance, the schema $H = [\ (\times\ Z\ *\ ),\ \ Z\ ,\ Z\ ]$ represents all the programs that include at least one occurrence of (× Z *) and at least two occurrences of Z. The tree fragment (× Z *) includes all the programs which have a × and a first argument of Z. Figure 4.3 shows an example for the instantiation of this schema.

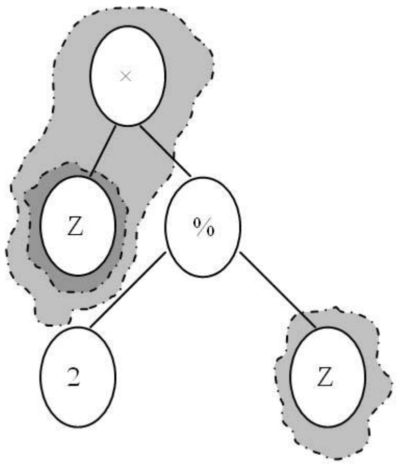


Figure 4.3 An example of O'Reilly's schemata

The order of the schema is defined as the number of non-* nodes in the fragments contained in the schema. The defining length is the number of the links in the expressions and tree fragments in the schema as well as the links that connect them together. The schema theorem derived is as follows:

$$\begin{aligned} E[i(H,g+1)] &\geq i(H,g)\frac{f(H,g)}{\overline{f}(g)}\times \\ &\quad (1-p_c \times \max_{j\in\Phi} p^d(H,j,g)) \\ &= i(H,g)\frac{f(H,g)}{\overline{f}(g)}\times p^d(H,g) \end{aligned} \tag{4.5}$$

,where

$p^d(H,j,g)$ is the probability of disruption of schema *H* contained in program *j*,

i(*H*,*g*) is the number of instances of *H* at generation *g*,

f(*H*,*g*) is the mean fitness of the instances of *H*.

The probability of disruption is the ratio between the defining length of *H* contained in program *j* and the total number of crossover locations in *j*. This probability depends on the size, shape and composition of the tree *j* matching the schema *H*. The mean fitness of the instances of *H* is calculated as the weighted sum of the fitnesses of the programs matching *H*. The weights are computed as the ratios between the number of instances of *H* in each program and the total number of instances of *H* in the population. The drawback of using the maximum probability is that it produces a conservative measure of the number of schemata in the next generation.

4.4.4 Rosca's Rooted Tree Schemata

A schema is a contiguous tree fragment, which includes the root node of the tree [274]. For instance, the rooted tree schema $H = (\times\ X\ *)$ represents all the programs with a root node of × and first argument of *X*. Figure 4.4 shows an example for the instantiation of this schema. This contains the positional information in the definition of the schema, which was absent in the previous three definitions. This implies that a schema *H* can be instantiated only once within a program.

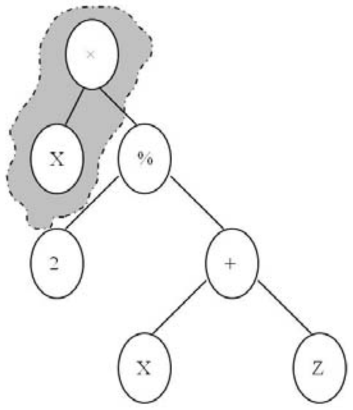


Figure 4.4 An example of Rosca's schemata

$$E[m(H,g+1)] \geq m(H,g)\frac{f(H,g)}{\bar{f}(g)}[1-(p_m+p_c)]\times \sum_{j\in H\cap\Pi}\frac{O(H)}{N(j)}\times\frac{f_j}{\sum_{j\in H\cap\Pi} f_j} \quad (4.6)$$

, where

N(*j*) is the size of the program *j* matching the schema *H*,

O(*H*) is the order of schema H, which is the number of defining symbols in *H*

Π is a multiset representing the population and $H\cap\Pi$ represents the multisets including all programs in Π which are also members of *H*.

4.4.5 Poli & Langdon's Schemata (Fixed Shape and Size Schemata)

A schema is a rooted tree composed of nodes from the combined set and * representing exactly a single function or terminal [257]. The operator * is a polymorphic function where its arity is the number of different arities from the combined set.

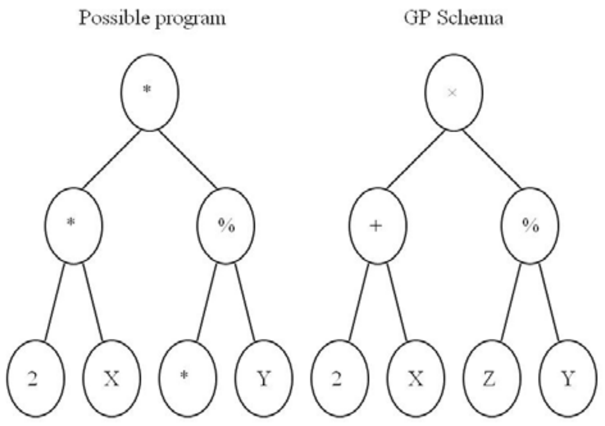


Figure 4.5 An example of Poli & Langdon schemata together with its program

The proposal in [257] was in line with the original schema definition for GA. The schema *H* represents multiple programs, all having the same labels for the non-* nodes and the same shape as the tree corresponding to *H*. The order *O*(*H*) of the schema *H* is defined as to the number of non-* symbols. The length *N*(*H*) of the schema *H* is defined as the total number of nodes in the schema. The defining length *L*(*H*) is the number of links in the minimum tree fragment that includes all the non-* symbols within the schema. This definition of schema also contains the positional information, which was absent in the other definitions.

Definition 4.4 The schema *G* is a **hyperspace** if it does not contain any defining nodes, i.e. *O*(*G*) = 0. Each hyperspace represents all the programs with a given shape.

Definition 4.5 The schema *H* is a **hyperplane** (ordinary schemata) if it contains at least one defining node.

Definition 4.6 The schema ***G*(*H*) is called the hyperspace associated with *H*** and is obtained by replacing all the defining nodes in a hyperplane H with *-symbols.

The one-point mutation and one-point crossover are defined in [257]. The one-point mutation involves the replacement of a terminal with another terminal and replacement of a function with another function with the same arity. The one-point crossover (further discussed in the next chapter) involves the alignments of parents and the selection of a common crossover point followed by the swapping of the two sub-trees below this point. For a GP with fitness proportionate selection, one-point mutation and one-point crossover the schema theorem can be stated as follows:

$$E[m(H, g+1)] \geq M \times p(H,g)(1-p_m)^{O(H)} \times \left\{1 - p_c\left[p_{diff}(g)(1-p(G(H),g)) + \frac{L(H)}{(N(H)-1)}(p(G(H),g) - p(h,g))\right]\right\} \quad (4.7)$$

, where

$p(H,g) = m(H,g)\dfrac{f(H,g)}{M\bar{f}(g)}$, is the probability of selection of schema *H*,

p_{m} is the probability that a location in the tree is mutated,

$p_{\mathrm{diff}}(g)$ is the probability that the offspring produced by 1-point crossover between programs *h* and $\hat{h}$ does not match *H*, given that *h* matches *H* and $\hat{h}$ does not match *H*.

4.4.6 Exact GP Schema Theorems

In the last few subsections, the pessimistic schema theories for GP were presented. These theories only consider the worst-case situation for schema disruption. Hence, they only provide the lower bounds for the expected number of instances of a schema *H* at generation *g*+1. It is required to develop theories that can model schema creation exactly, rather than providing a lower bound for the expected number.

Definition 4.7 The ***total schema transmission probability*** *for the schema H,* ***α(H,g)*** *,* represents the probability that at generation *g* the trees produced will match *H.*

If the programs all have exactly the same shape and size and use the one-point crossover operator, then the total transmission probability of a schema *H* can be expressed as:

$$\alpha(H,g) = (1-p_c)p(H,g) + \frac{p_c}{N(H)}\sum_{k=0}^{N(H)-1} p(l(H,k),g)\,p(u(H,k),g) \quad (4.8)$$

, where

l(*H*,*k*) is the schema obtained by replacing all of the nodes above point *k* with the don't care symbol (the symbol *l* stands for "lower part of"),

u(*H*,*k*) is the schema obtained by replacing all of the nodes below point *k* with the don't care symbol (the symbol *u* stands for "upper part of"),

Definition 4.8	The ***hyperschema function set*** is the function set in GP and the *-symbol. The * symbol represents only one node.
Definition 4.9	The ***hyperschema terminal set*** is the terminal set in GP plus the *-symbol and the #-symbol. The #-symbol represents any valid sub-tree
Definition 4.10	The ***GP hyperschema*** is a rooted tree composed from the hyperschema function set and hyperschema terminal set.

The generalisation of (4.8) is valid for individuals that include variable sizes and shapes and is given by (4.9).

$$\alpha(H,g) = (1-p_c)p(H,g) + \\ p_c \sum_h \sum_{\hat{h}} \frac{p(g,g)p(\hat{h},g)}{NC(h,\hat{h})} \sum_{k \in C(h,\hat{h})} \delta(h \in L(H,k))\delta(\hat{h} \in U(H,k)) \qquad (4.9)$$

, where,

$NC(h,\hat{h})$ is the number of nodes representing the common region between the program h and the program $\hat{h}$,

$C(h,\hat{h})$ is the set of indices of the crossover points in the common region,

$\delta(x)$ returns 1 if *x* is *TRUE*, otherwise 0,

L(*H*,*k*) is the hyperschema produced by replacing the nodes between the crossover point *k* and the root node with * and all the subtrees connected to those nodes with #,

U(*H*,*k*) is the hyperschema produced by replacing the subtree below the crossover point *k* with # and,

the first two summations in (4.9) are over all the individuals in the population.

4.5 Concluding Remarks

In this chapter, some of the theoretical basics in the field of GP were presented. A major part of this chapter introduced the concept of schema in GP. The schema theory may provide a better understanding of the dynamics of the populations in GP. It can predict the expected number of instances of a schema in the next generation. It provides a way of modelling the algorithm, its representation and operators. However, it should be noted that schema theorems have not been fully developed or exploited [195] and therefore a more extensive research in this area is required to further increase our understandings of the theory of GP and possibly pave the way to new innovative directions.

CHAPTER 5

A SURVEY AND TAXONOMY OF PERFORMANCE IMPROVEMENT OF GP

"If I have seen further it is by standing on the shoulders of giants", Sir Isaac Newton.

5.1 Classification of Evolutionary Algorithms

In the natural world, there is a wealth of complex and intelligent biological organisms and creatures. Consequently, researchers have viewed evolutionary processes as a powerful concept, resulting in the birth of Evolutionary Computation (EC). As previously described in Chapter one and depicted in Figure 5.1, Evolutionary Computation has four main traditional variants; Evolutionary Programming (EP), Evolution Strategies (ES), Genetic Algorithms (GA) and the youngest stream the Genetic Programming (GP). All the EC variants are based on the same concept of Darwinian evolution and thus the underlying idea is the same for all of them. That is, they start off with an initial random population of individuals and process this set of candidate solutions simultaneously, using natural selection and operations such as crossover and mutation to produce new candidate solutions. All the various dialects of EC are population based and stochastic. They use random initialisation together with architecture altering operations, fitness function, selection mechanisms and termination conditions to discover a solution.

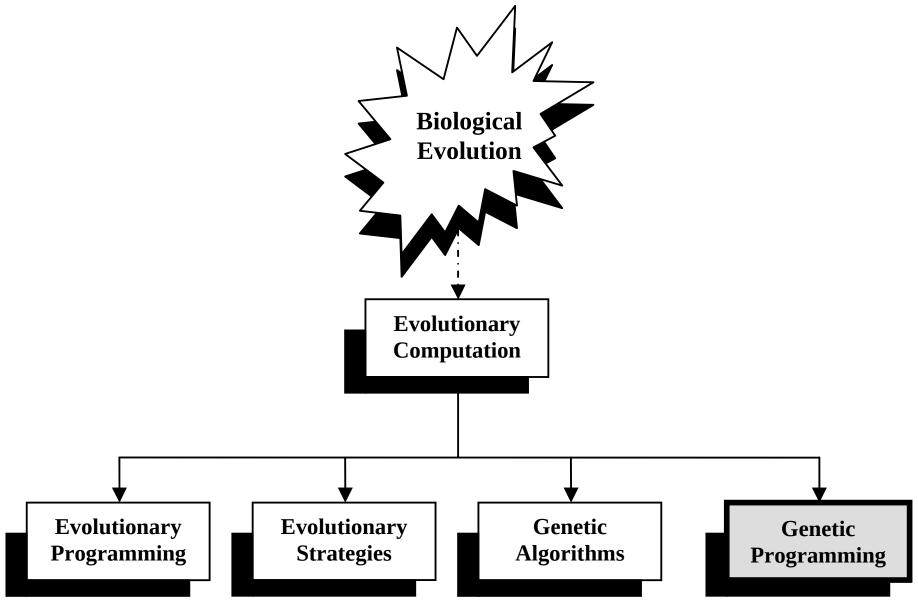


Figure 5.1 Evolutionary Computation with its main variants

The primary feature that characterises each EC system into its own stream is how the chromosomes are encoded. In fact, the different EC systems were organised in [13] based on their type of representation. This point is graphically illustrated in Figure 5.2. In other words, the main difference is in the structure undergoing adaptation, i.e. the representation or the genotypes. Consequently, the definition of their respective variational operators becomes specific and different. Each discipline therefore differs in its application area.

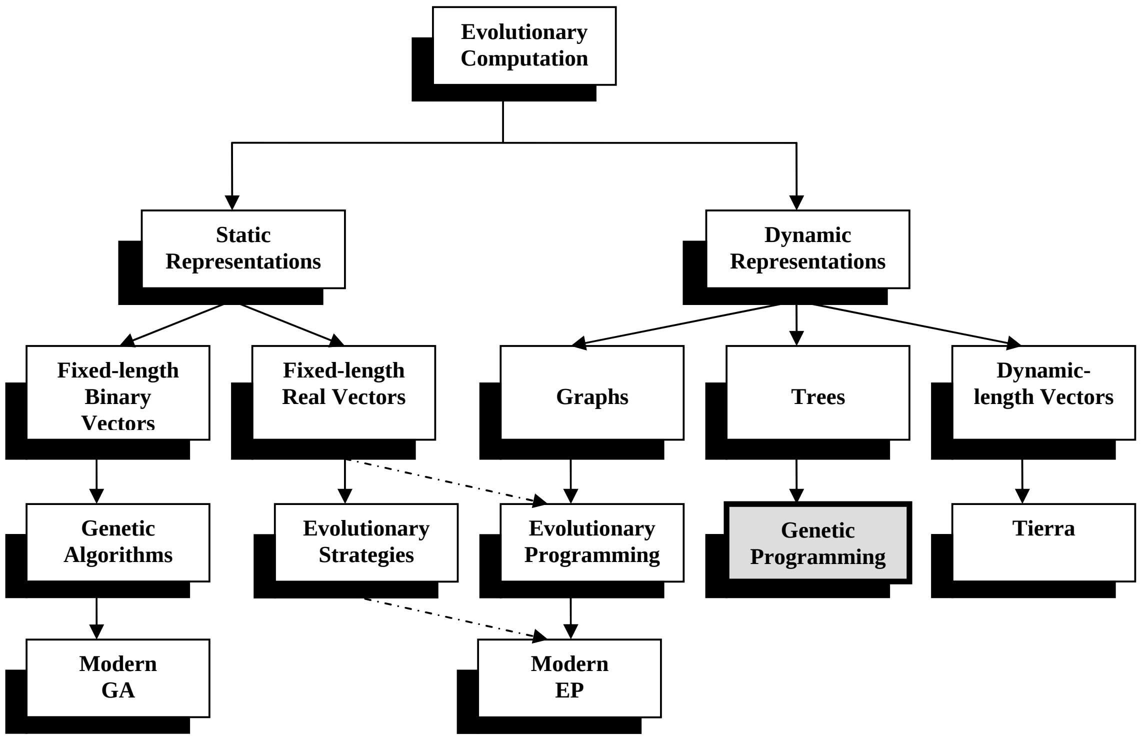


Figure 5.2 Alternate taxonomy based on type of representation [13]

This chapter presents a comprehensive overview and taxonomy of previous research conducted to enhance the performance of Genetic Programming. Although this chapter covers other variants of the Genetic Programming paradigm, its main focus is on the canonical or standard GP which was championed by Koza in [167] [170] based on tree structures. The creative ideas proposed by researchers to improve the effectiveness and efficiency of GP are reviewed and categorised in the following sections. Section 5.2 discusses the overall modifications and improvements. Section 5.3 looks at Improved GP. Sections 5.4 to 5.6 discuss the proposed modifications that various researchers have put forward and form the bulk of this chapter. In Section 5.7, some of the GP variants and Hybrids are highlighted and briefly discussed.

5.2 Overview of Modifications and Improvements

The modifications that have been made to the GP to improve performance or save computational effort generally fall into three categories [166], GP variants, hybrids and improved GP as depicted in Figure 5.3 and described below.

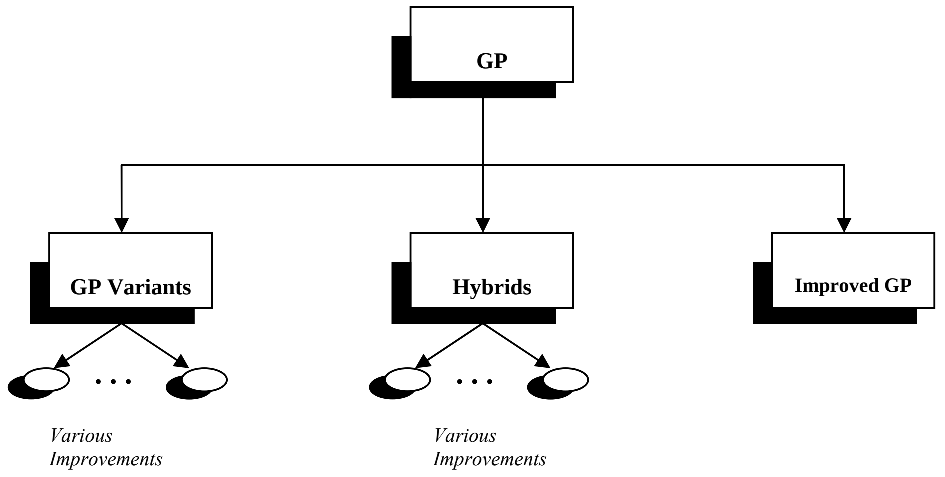


Figure 5.3 Variants of canonical GP

The structures within the canonical or standard GP are tree-based. However, the structures that undergo adaptation within the GP variants are no longer tree-based and significantly deviate from the original GP championed by Koza. As previously mentioned, since the genome or structure is the primary feature that distinguishes the different variants within EC, the same argument can be applied here. In other words, these variants of GP can be viewed as separate minor dialects of EC because the representation is changed but they have the same motivation or application of a GP. Hybridisation of EC with other techniques, generally known as Memetic Algorithms (MA) [71], are considered to be problem tailored methods. The combination of GP with other algorithms or problem specific techniques enriches the GP with knowledge and thereby improves its performance. All the other innovative methods to improve the performance of GP fall in the last category, termed here as Improved GP (IGP), which preserves the tree-based structure of the canonical GP.

Although GP Variants and Hybrids are briefly discussed herein, the focus will be based on Improved GP.

5.3 Improved GP

Figure 5.4 shows how the various improvements on Standard GP may be organised into possible categories. Generally the improvements are either implemented to remedy an acknowledged issue within Genetic Programming, such as the bloat phenomenon, lack of diversity or premature convergence etc, or the improvements provide pioneering modifications to the components of the GP algorithm. The proposed modifications can usually be further subdivided into three classes. They could be fixed and predetermined prior to the run and left unaltered during the run. For example, one may determine a certain variation operation is beneficial and recommend its use for the entire run. Secondly, the modifications could be deterministic and static - there is a predetermined rule which specifies how and when a certain modification will take place. The rule is a function $f(g)$ of time or generation g. For example, it may be suggested that the mutation operation should only be used after generation number $g>30$. Lastly, the modifications could be based on some feedback mechanism. For example, if a certain condition occurs, then a particular selection mechanism or operation should take effect.

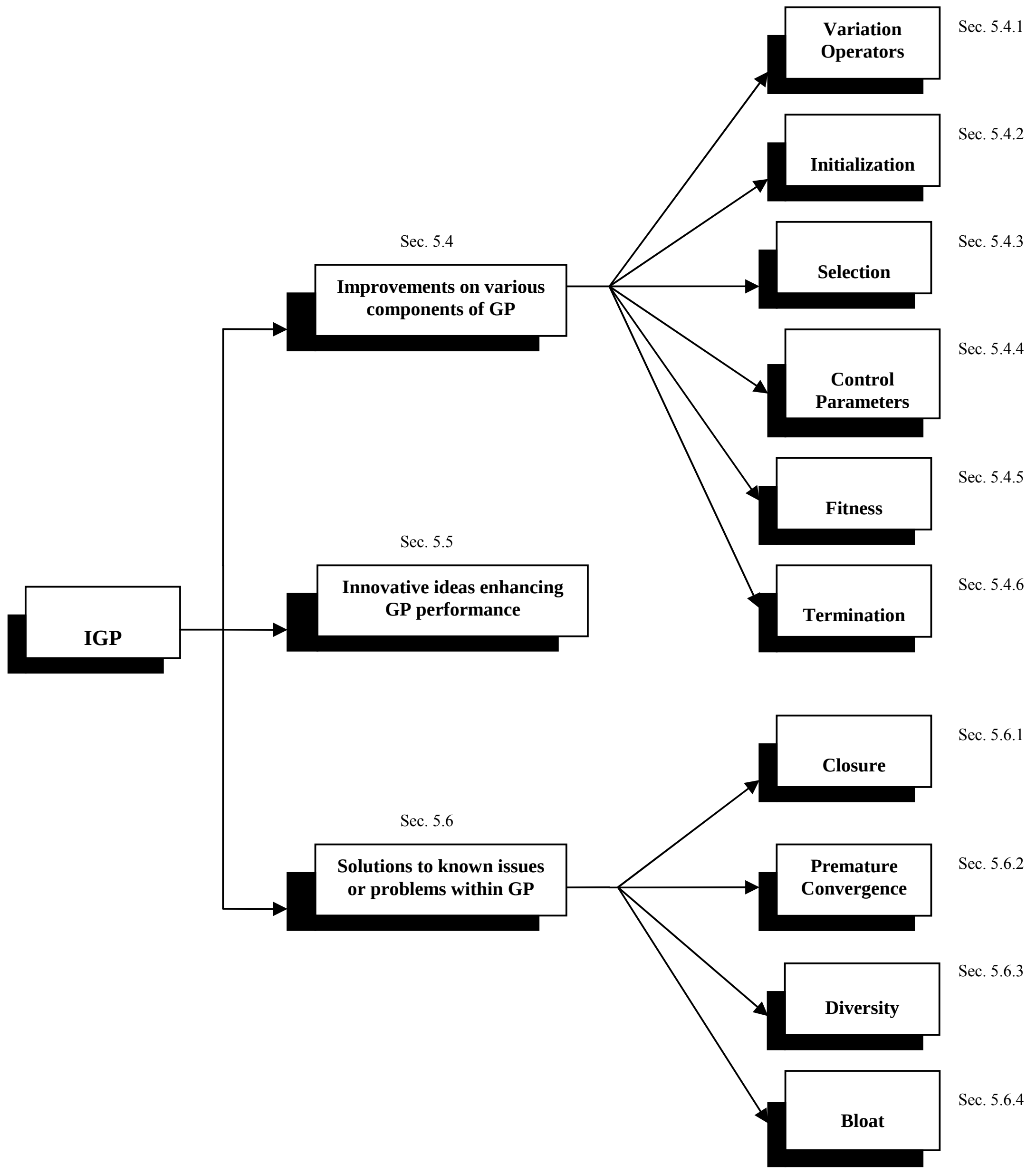


Figure 5.4 Classification of IGP into different categories

5.4 Improvements on Components of GP

Genetic Programming has a number of main components which define its operation, namely variation operators, initialisation, selection, control parameters, fitness and termination. Researchers have looked into improving or altering all these components in turn to enhance the performance of GP, as detailed in the following subsections.

5.4.1 Variation Operator

Variation operators are used to create new candidate solutions and are typically divided based on their arity, e.g. mutation and crossover being unary and binary variation operators respectively. Although crossover shoulders a great responsibility for the evolution of the GP algorithm [74][75], the mutation operator plays a significant role in the genetic convergence process by preventing loss of genetic diversity in the population [197]. The main search operator in GP is the crossover operator and all the other variation operators are often considered as secondary operations. The crossover operator is discussed in the following sub-subsection and the next subsection reviews the secondary operators and all the other newly proposed operators.

5.4.1.1 Crossover

The primary operation for modifying genetic structures in GP is crossover. Crossover is a stochastic operator which merges information from generally two parents to create offspring genotypes. The first crossover operator for GP was defined in [167]. The effect of crossover was investigated in [243]. It was argued that it has the disadvantage of producing a high computational cost due to growth of individuals in size and complexity during the evolution process [303]. This effect, which is known as code bloat, is formed by an excessive exploration capability of the crossover [217]. It was argued [21] that 75% of crossover event could be termed as lethal and can result in disruption of building blocks. With recombination or crossover being considered as the primary operator in GP, many researchers have looked into ways of modifying it to improve the efficiency.

As crossover was viewed to be destructive, the brood recombination aimed to decrease this effect and preserve good building blocks. The "soft brood selection" method [1] generated a brood by performing crossover over the selected parents *N* times and then introduced the best of the brood in the next generation by holding a tournament. The "brood recombination" was introduced in [298] [299], which was a refinement of the soft brood selection. The Brood Selection Recombination Operator $R_B(n)$ produced n pairs of offspring but only kept the best two of the $2\times n$ produced offspring using a selection function. As the brood selection performs multiple samplings of the crossover operator and keeps the best 2 offspring, it can essentially be viewed as a hill-climbing crossover operator. Although, the brood selection increases the computational cost per generation it can however increase the selection pressure and therefore the rate of convergence towards an optimal solution for some problems. To reduce the computational cost a clever approach was implemented [298], where the evaluation of the new $2\times n$ offspring is on a small portion of the training set rather than all the test cases. The brood size was further investigated [337] for the brood recombination crossover method. It was shown that as the brood size increased, the performance improved and the brood recombination method outperformed the standard crossover method for the three object classification problems studied. The disruptive nature of crossover was reduced by the brood recombination, as the children of the destructive crossover events were rejected by this operator and as a result the building of larger building blocks was promoted.

Other approaches [259][332] choose good sub-trees or crossover points to swap. Context-Aware Crossover [139][216] discovers the best possible crossover site for a sub-tree and is shown to consistently attain higher fitness. There are similarities between the context-aware crossover and the Brood Crossover in that multiple children are produced during each crossover event. The destructive effects of standard crossover was minimised in [214] by placing the selected sub-tree in its best context in the parent tree. The best context was calculated by using the effect of the placement of the selected sub-tree on the overall fitness of the parent tree and then selecting the placement, which produced the maximum final fitness.

Some heuristics were added to the standard crossover operator [131] to make it smart. Here, a form of intelligent heuristic guidance for the GP crossover was proposed. The smart crossover computed the performance values for sub-trees and used this information to decide which sub-trees are potential building blocks to be inserted into another sub-tree and which sub-trees are to be replaced due to their poor performance value.

A homologous crossover operator was introduced in [21], where the exchange is strongly biased towards very similar chunks of genome. Structural distances are measured by comparing genotypes and functional distances by comparing phenotypes. These two measures are used to determine the probability that the trees are crossed over at a specific node. In this way, the crossover probabilities are biased by structural and functional features of the trees. Similarly, a GP 1-Point crossover operator was introduced in [256], which had homologous overtones. This was based on the one-point crossover for GAs, where the selection process involved checking for structural similarities of trees to find points with structural homology.

The Ripple Crossover, examined in [152] , was shown to outperform the traditional sub-tree crossover on two benchmark problems. Although the Ripple Crossover was more explorative than the single tree node crossover, it was a more disruptive crossover operator and its disruptive nature resulted in a slower convergence. A one-point crossover was introduced in [257], in which the same crossover point is selected in both parents. Two trees are aligned from the root nodes and recursively and jointly traversed. Recursion is stopped as soon as an arity mismatch between the corresponding nodes in the two trees is observed. A random crossover point is selected from the above identified nodes and the two sub-trees below the common crossover point are swapped. Some of the interesting features of the one-point crossover are that it is a simpler form of crossover for GP and it facilitates population convergence by searching for good partial upper part or structure solutions. Moreover, it does not increase the depth of the offspring beyond that of their parents.

Crossover points are conventionally selected randomly. A depth-dependent crossover for GP was proposed [123], in which the depth selection ratio was varied according to the depth of a node. Shallow nodes were favoured as the crossover points and hence larger sub-trees were swapped. This promoted the accumulation of useful building blocks via the encapsulation of a larger part of a tree. The behaviour of the uniform crossover and point mutation was examined in [251] presenting a novel representation of function nodes, which allowed the search operators to make smaller movements around the solution space. It was shown that the performance on the even-6-parity problem was improved by three orders of magnitude when compared with the standard GP.

The headless chicken crossover operator, which was studied in [188], uses a selected program *P* and a newly randomly generated program *R* to produce an offspring by replacing a sub-tree of *P* with a replaced sub-tree from *R* until it finds an offspring with greater or equal fitness to *P*. The crossover-hill climbing [21] operator is another form of the headless chicken crossover. In [3] [114], the fitness of the individual is the sum of its fitness components or genes. A gene is periodically added to the individual during the evolution and if it improves its fitness it is kept, otherwise discarded. Between gene additions, the population evolves by intergene crossover.

A novel crossover method was proposed [150] using the usage frequency of nodes. Three crossover techniques were investigated, namely a crossover with crossover points in both nodes having high usage frequency, with crossover points in a node having high usage frequency and a node having low usage frequency, with crossover points in both nodes having low usage frequency. It was discovered that their method was promising for speedup in GP. Many researchers looked into determining crossover points that are likely to be more advantageous. For example, a higher-level analysis of the population as a whole was used in [271] utilizing statistics gathered over all sub-trees to determine the crossover points or in [15] Selective Self-Adaptive Crossover (SSAC) and Self-Adaptive Multi-Crossover (SAMC) methods were used. The depth-fair crossover (DFC) was introduced in [156], which allowed for weighting crossover points. It assigned an equal weight to each depth of the tree. Each node within each depth was given an equal amount of the depth weight. Improvements were also made to the crossover operator [338] using a measure called looseness to guide the selection of crossover points rather than choosing them randomly. Improvement was shown over the headless chicken crossover [188] and the standard crossover.

The latest developments imply that the crossover operator is on its way to becoming a more powerful and robust operator. It is believed that there is still room for the crossover operator to improve the quality and efficiency of the search it conducts. This can be achieved by either combining the current approaches or devising new ways for improvement.

5.4.1.2 Other Operators

In addition to the primary genetic operation of crossover, there are various secondary operations such as mutation, permutation encapsulation etc. The effect of various operators, namely mutation, permutation, encapsulation and editing, on GP performance was first investigated in [170]. Although it was argued that the subject was certainly not solved and required further work for a general conclusion, it was shown that for some selected problems there was no substantial difference in performance when these operators were included.

The performance improvements in GP provided by Automatic Defined Functions (ADF) and decimation were compared in [235] using the Santa Fe ant, the lawnmower, the even 3-bit parity and symbolic regression problems. It was concluded that decimation provided superior improvement in performance over ADF. It should however be noted that it was concluded that ADF was not effective for simple problems [174] and its benefits only became increasingly evident for complex problems.

To overcome the disruption of building-blocks due to crossover and mutation, an adaptive program called "STROGANOFF" (STructured Representation On Genetic Algorithms for Non-linear Function Fitting) was introduced [128] [129]. In [130] an adaptive recombination for a numerical GP was proposed which was guided by a measure called Minimum Description Length (MDL). The application of mutation or crossover operators was adaptively controlled to improve efficiency.

A new crossover operator was introduced in [74] [75] to minimise the number of evaluations required to find an ideal solution by evaluating the observed strengths and weaknesses of selected individuals within areas of the problem. The motivation in [75] was to intelligently perform crossover by discriminating between the portions of each parent that lead to success and failure. A new GP operator, the memetic crossover, was introduced, which allowed for an intelligent search of the feature-space. The proposed process involved the identification of specific areas of importance within the problem (sub-problem) and in tracking the nodes executed while observing the individual's performance as it was evaluated. The information gathered was then organised by ranking the nodes. Nodes that were executed were said to participate in the sub-problem. Bad and good nodes were associated with one or more sub-problem failures and successes respectively. Using the memetic crossover method [75] individuals were examined to ensure compatibility with respect to sub-problem performance. The individuals were regarded as compatible, when a significant sub-problem match occurred between one of the worst performing nodes in the recipient and one of the best performing nodes in a potential donor. In this instance, crossover was performed with the recipient replacing its bad node with the donor's good node. The Los Altos trail and the royal tree problem, which can easily be decomposed into well-defined sub-problems, were used as benchmark problems. For this approach to be significantly advantageous, it was required that the problem be able to be methodically decomposed into sub-problems. Although memetic crossover incurred additional processing cost, they were considered negligible when compared to the time saved through the reduction in the number of evaluations.

The macro-mutation operator (headless chicken crossover) was shown in [188] to outperform the traditional GP crossover operator. The pruning genetic operator was proposed in [239] for removing useless structures from the GP individual. The operation was applied to randomly selected sub-trees. Redundant node patterns, which are problem dependant and uniquely defined for each problem, were searched and replaced with an effective terminal node resulting in efficiency improvement in the GP's search.

In [19], crossover was between a member of the population and an ancestor tree, which was a fixed collection of trees. The crossover operator generated only one tree, the population member with one of its sub-trees replaced by a sub-tree of the ancestor. This variation operator, which was neither a crossover nor a mutation, used information from two individuals with only one member belonging to the population. Analysis of mean tree size growth demonstrated that this operation limited parse tree growth because the ancestors did not grow. The genetic material in the ancestor set did not change and was available indefinitely, implying that the building blocks or information was never lost.

It is suggested that new possible operators or methodologies should be devised that either promote finding good building blocks or reduce the destructive nature of the currently proposed GP operators. Care should however be exercised that that the reduction of disruptive effects of current operators, that generate new candidate solutions, should not be overindulged as it may simply transform the GP paradigm to a simple hill climber, which is not desirable.

5.4.2 Initialisation

Initialisation involves the random generation of individuals for the initial population. The traditional GP tree-creation algorithms GROW, FULL and Ramped Half-and-Half were introduced in [170]. It was shown that Ramped Half-and-Half was the best as observed for the Quartic polynomial, 6-multiplexer, Artificial Ant and Linear equations problems. In [194], a random initialisation, which produced programs of random shapes, was defined.

The RAND_tree algorithm was introduced in [132] and in [30] where trees were initialised with exact uniform probability from a tree-derivation grammar. Using diverse random seeds, multiple abbreviated runs were made in [254]. An enriched population was created using the best member from each abbreviated run. This enriched population was then loaded together with a full set of randomly generated unique members at the start of a consolidated run.

Two new tree-creation algorithms Probabilistic Tree-Creation (PTC 1 and PTC 2) were offered by [209], where an average tree size or a distribution of tree sizes could be specified with guaranteed probabilities of occurrence for specific terminal and non-terminal functions within the generated trees. PTC 1 & 2 had very low computational complexity and had comparable results with the GROW technique.

Further research in formulating novel ways of generating new individuals per run should be called for, as it is believed that the overall fitness and diversity of the initial population in the first generation plays a significant role in the success of the later generations within that run. The main goal should be to enrich the initial population and increase its structural and behavioural diversity without introducing a large computational effort. Finer initialisation techniques for new runs may emerge by either exploring memory or learned behaviour from previous runs or introducing problem specific innovations into the initialisation stage.

5.4.3 Selection

The role of selection is to differentiate among individuals based on their quality. Individuals with higher quality are favoured to be parents participating in a variation operation or be replacements of an existing individual in the case of recombination. Selection is generally responsible for driving quality improvements and is probabilistic. The performance characteristics of a repertoire of selection methods, namely proportional selection, ranking selection, and tournament selection, were investigated for time series prediction in [157].

In [260], the sampling behaviour of tournament selection over multiple generations was analysed, where the analysis was focused on individuals which did not participate in any tournament at all, due to not being sampled during the creation of the required tournament sets. A new selection scheme was proposed in [92], which was based on standard tournament selection, to encourage genetically dissimilar individuals to undergo genetic operation. It demonstrated performance improvements of GP for the algebraic symbolic regression problem. An automatic selection pressure for the tournament selection was investigated in [327] to improve the efficiency of GP. The number of tournament candidates was dynamically changed in response to the changing evolutionary process. Using the symbolic regression and the even-6 parity problems it was shown that this approach could improve the effectiveness and efficiency of GP systems. In [330], the relationship between population size and tournament size was investigated and a new fitness evaluation saving algorithm, Evaluated-just-in-time (Ejit), was proposed which resulted in constant computational savings by avoiding the evaluation of not-sampled individuals.

It is believed that a large training set of fitness cases could slow down GP. Dynamic subset selection based on a fitness case was proposed in [196], where an appropriate topology-based subset selection method allowed individuals to be evaluated on a smaller subset of fitness cases. A topology relationship on the set of fitness cases was created during the evolutionary search by increasing the strength of the relation between two fitness cases that an individual could successfully solve. The proposed selection method selected a subset, where the fitness cases were distantly related with respect to the induced topology. Using four different problems, it was shown that dynamic topology-based selection of fitness cases progressed on average faster than the stochastic subset sampling.

In the canonical GP, selection pressure is only applied in the selection of parents and the offspring are simply propagated into the next generation without any selection. In [329], a many-offspring breeding process with selection pressure applied to the selection of offspring was investigated. A many-offspring breeding process can be viewed as a standard crossover that generates a large number of poor offspring in the search for good offspring. Two crossover operators were proposed. Firstly, the Ideal Crossover considers all the possible ways of recombining two selected parents to produce all the possible offspring. It then evaluates all the offspring and keeps the best two offspring with the highest fitness values. Secondly, the Partial Crossover, which is similar to the context-aware crossover operator [214], selects a random point for crossover in one parent P_1 but considers all the other nodes in the other parent P_2 to produce offspring. The focus of these techniques is to optimise the offspring's fitness and thereby increasing selection pressure.

A theoretical and empirical study that will increase our understanding of which selection methodologies can be deemed superior during different stages of the evolutionary process is recommended. This study can then become the basis for implementing new schemes that explore dynamic selection of various proposed selection techniques during the run. In addition, further innovative ways that can result in reducing the computational burden of selection can be very beneficial for enhancing the performance of GP.

5.4.4 Control Parameters

The genetic programming paradigm is controlled by various control parameters such as the maximum number of generations (*G*), the population size (*M*), the probability of crossover (P_c), recombination (P_r), mutation (P_m) and the maximum tree depth (*D*), to name but a few.

The issue of parameter control and setting was discussed in [71]. The algorithm parameters can either be tuned or adapted. Parameter tuning involves the empirical investigation of parameter values which will result in good performance before the run. Once the best suited parameter value for the specific problem is determined, its value is set in advance and remains unchanged for the duration of the run. Alternatively, the parameters could be deterministically altered as a function of time/generation during the run or adaptively controlled through some heuristic feedback mechanism resulting in explicit adaptation. On the other hand, the actual parameters could be encoded into the data structures of the algorithm and evolve with the adaptation being entirely implicit. This can be summarised as per Figure 5.5.

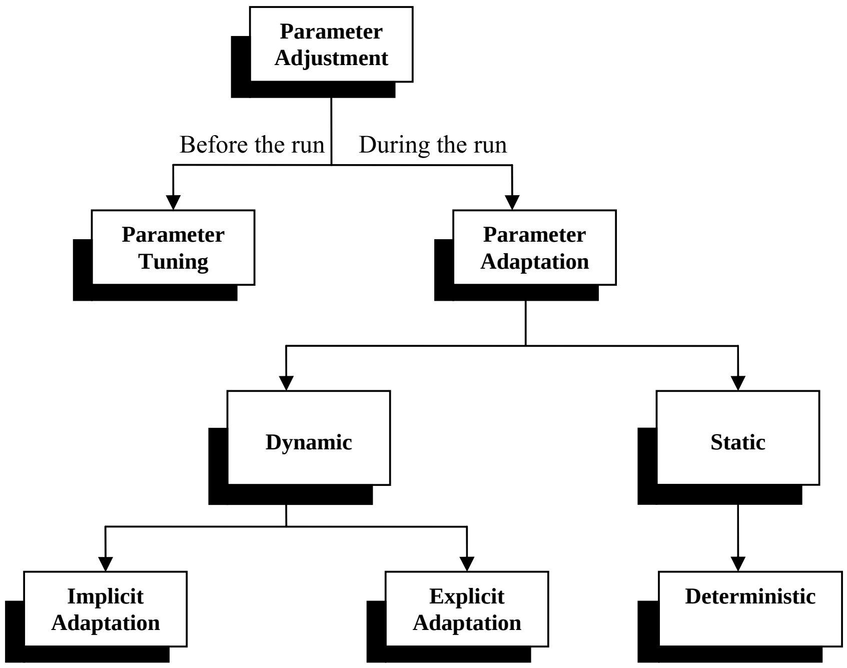


Figure 5.5 Taxonomy of parameter adjustment

5.4.4.1 Tree Size

There has been substantial amount of work performed concerning the solution's shape (dynamics of tree size and depth) such as [53][54][65][177][288], to name but a few. The tree size parameter is used to impose a size restriction on individuals. Typically the tree sizes for the random initial population and evolved individuals are restricted differently, namely through the maximum initial tree size D_i and the maximum created initial size D_c. These parameters impose restrictions on the maximum allowable depth for a tree. The correlation between average parent tree size and the modification point (crossover or mutation) was shown in [213]. Both of these were directly linked to the size of the resulting child.

A dynamic tree depth limit was explored in [283] [284]. The dynamic limit was initially set as high as the maximum depth of the initial random trees [283]. Trees which exceeded this threshold were rejected and replaced by one of their parents, unless the tree in question was the best individual found so far. In this instance, the dynamic limit was increased to match the depth of this new *best-of-run* individual. The dynamic limit was lowered as the new *best-of-run* individual allowed such reduction [284]. Moreover, a dynamic tree depth was adopted in [342] to constrain the complexity of programs. The proposed method was applied to data fitting and forecasting problems with results indicating improvement over GP.

Some researchers have looked into limiting the total amount of tree nodes of the entire population [311], rather than imposing limits at the individual level. The concept of resource-limited GP, which was a further development to [311], was introduced in [285]. As the total number of nodes in the population exceeded a predefined limit, resources became scarce and not all offspring were guaranteed to progress. The candidates were queued by fitness and progressed into the next generation on a first come first serve basis. The trees that required more resources than the amount still available would not survive. The relationship between size and fitness was not explicitly defined and was a product of the evolutionary process. A natural side effect of this approach was that the population was automatically resized. Although these approaches used the same rationale, they operated at different levels of the GP paradigm, namely acting at the individual level and at the population level. Tree depth limits imposed a maximum depth to each individual and Resource-limited GP limited the total amount of resources that the entire population could use.

The two different approaches, tree depth limits and resource-limited GP, were compared in [286] using symbolic regression, even parity, and artificial ant problems. It was shown that the resource-limited GP was superior to tree depth limits [286]. A dynamic approach to resource-limited GP was developed in [287]. The dynamic resource limit was initially set as high as the amount of resources required for the first generation. The allocation of trees, sorted according to fitness, continued into the next generation until the resources were exhausted (as per the original resource-limited GP). The rejected individuals would be considered as candidates for the next generation if the mean population fitness was improved. Hence the dynamic resource limit was raised as a function of mean population fitness providing the additionally needed resources. It was shown that the dynamic approach to resource-limited GP achieved better performance when compared with the static approach and with the traditional depth limits, using the symbolic regression polynomial problem and Santa Fe artificial ant problem.

5.4.4.2 Operator and Selection Probabilities

As GP is a completely stochastic process, it is controlled by various probabilistic control parameters, such as the probability of selecting an internal point (P_{ip}) as a node for the crossover operation or probability of crossover and reproduction (P_{c} and P_{r}) which determine by which process the fraction of individuals are created for the next generation.

In [243], Explicitly Defined Introns (EDIs) were introduced as instruction segments that were inserted between two nodes of useful code. EDIs changed the probability of crossover between the two nodes on either side of the EDI improving the convergence properties of a GP algorithm. Similarly in [38] the probability of selection of every node for crossover was indirectly adapted through the evolutive introns (EIs). Evolutive introns are explicitly defined introns, which are artificially generated, with the aim to increase the probability of selecting good crossover points as the evolutionary process continues. The automatic growth and shrinking of non-coding segments in the individuals are promoted, thereby adapting the probabilities of groups of code being protected.

The adaptation of operator probabilities in genetic programming was investigated in [237] with an attempt to reduce the number of free parameters within GP. Two problems from the areas of symbolic regression and classification were used to show that the results were better than randomly chosen parameter sets and could contest with parameters set as based on empirical knowledge.

5.4.4.3 Terminal and Function Sets

In the conventional GP, the structures are typically comprised of the set of N_{func} functions from the function set $F = \{f_1, f_2, \cdots, f_{Nfunc}\}$ and the set of N_{term} terminals from the terminal set $T = \{a_1, a_2, \cdots, a_{Nterm}\}$ forming the combined set $C = F \cup T$. This combined set defines the set of all the possible structures or elements. The choice of function and terminal sets can have a significant effect on the GP's performance and if the sets are not sufficient to express a solution for a given problem, then GP would not be able to solve the problem.

The effect of extraneous variables and functions was first studied by [170] and it was shown that a linear degradation in performance was observed as additional extraneous variables were added for the cubic polynomial problem. Similar results were obtained for extraneous functions; nevertheless it was shown that for some specific problems, extraneous functions improved performance. Furthermore, no substantial difference in performance was observed for extraneous ephemeral random constants. It was concluded that the question of extraneous sets was not definitely answered in this study and further experimental and theoretical work was recommended to be carried out to lead to general conclusions. In [313], a systematic study was conducted of how to select appropriate function sets to optimise performance. They classified functions into function groups of equivalent functions and showed that a set that was optimally diverse (that included one function from each function group) was most appropriate.

5.4.4.4 Population and Generation Number

Population size (M) and the maximum number of generations (G) are the two major numerical control parameters in GP, with their values generally dependant on the difficulty of the problem. Dynamic and adaptive population sizing has been previously studied in the areas of genetic algorithms and particle swarm optimisation [122] [205] [208] [300]. Some controversy has been reported in the past literature on whether on-the-fly population sizing is effective [72] [206] in EA. The role of population size in GP was *first* very briefly studied by [170], with the population size maintained at a constant level and not varied throughout the run. It was shown that using a large population was not always the best way to solve problems [93]. The control of population size in GP was first implemented in [292] to improve the algorithm's robustness and reliability. The plague operator (first experimented with in GA [62] [32]) was introduced in [79] to fight bloat in GP, where individuals were removed at a linear rate to compensate for the increase in individual size. The decrease in population size was also studied in [212], where the population size was gradually decreased throughout the GP run. In Virtual Ramping the size of the population and the number of generations were continuously increased [83] reducing premature convergence. In [164], the population variation scheme was introduced. Fernandez *et al.* [81] introduced a technique to dynamically vary the size of the population during the execution of the GP system. In [165] various new ways to dynamically vary the population size during the run of the GP system were proposed. It should be noted that population implosion [212] is the same as the plague operator, as far as this thesis is concerned. They both reduce the population size at a linear rate. Their only difference is in the way they measure computational effort. In plague, the number of nodes are counted (as the main intention is reduction of bloat) while reducing population size, whereas in population implosion, the number of evaluations are counted. Therefore, as far as this thesis is concerned both works are identical in the way they reduce population size, and their names will be used interchangeably.

5.4.5 Fitness and Objective Function

Fitness is the driving force of natural selection and measures the quality of an individual with respect to how well an individual can solve a given problem. It is generally defined by an objective or fitness function forming the basis for selection and defining and facilitating improvements.

Multi-objective techniques [39], which allow the concurrent optimisation of several objectives by searching the so-called Pareto-optimal solutions, were investigated in [27] to evolve compact programs. There are various multi-objective optimisation techniques such as SPEA (Strength Pareto Evolutionary Algorithm) [343], SPEA2 (an improved version of SPEA) [345], SPEA2+ [158], NSGA-11 [64], Adaptive Parsimony Pressure [334]. The program size was considered to be a second independent objective in addition to the program functionality in [27] and an enhanced version of SPEA proposed in [344] was used. A multi-objective GP was used in [18] to make improvements on the results.

Novel strategies based on elastic artificial selection (EAS) and improved minimum description length (IMDL) were investigated in [162] for fitness evaluation and selection to create shorter programs and prevent premature convergence. The effect of tournament selection and fitness proportionate selection with and without over-selection for particular problems was investigated in [170]. It was shown that for many problems it was possible to enhance the performance of GP by greedily over-selecting the fitter individuals in the population.

In [328] the whole population was clustered to reduce the fitness evaluations and improve the effectiveness of GP. The clustering was performed by a heuristic called fitness-case-equivalence. For each cluster, a cluster representative was selected and its fitness calculated and directly assigned to other members in the same cluster. Using a clustering tournament selection method and a series of experiments of symbolic regression, binary classification and multi-class classification problems, it was shown that the new GP system outperformed the standard GP on these problems.

A new methodology was proposed [233] to create a new training set of randomly-generated fitness cases prior to each generation of the GP run instead of using a fixed set of fitness cases. It was shown that, this methodology was mainly useful to reduce the brittleness of GP when the fixed training population does not adequately represent the full range of difficult situations of the problem. The fitness function was scaled over time in order to improve performance [94]. The motivation behind this approach was that it is often easier to learn difficult tasks after simpler tasks have been learned.

As the majority of the computational effort in GP is expended in the fitness function, it is beneficial to avoid invoking the fitness function whenever possible. As the reproduction operator produces an identical copy of its parent, it is then quite obvious that the fitness evaluation can be safely avoided. This can result in considerable savings in computation because reproduction in GP conventionally accounts for the creation of ten percent of the new individuals. This flagging and caching of the already computed fitness of reproduced individuals was proposed in [170], provided that there are no varying fitness cases from generation to generation. A technique was devised in [140], which allowed the GP system to determine many instances in which invocation of the fitness function could be avoided. This was achieved through the consideration of the program nodes executed during fitness evaluation to establish whether a newly generated individual has the same fitness value as its parent. This could be realised through the identification of dormant nodes, which are program nodes that are never executed, extending a marking method described in [28]. It was shown that this technique, when applied to the multiplexer problem and even-parity problem, resulted in significant savings in execution time.

5.4.6 Termination

In traditional GP, a fixed number of generations are usually used as the condition for terminating the evolution process. To address the CPU time-consuming issue and the large amount of computational resources required for GP, the three different termination criteria of effort, time and max-generation were examined [89]. An improved termination criterion was implemented in [186] to prevent premature termination, when further search may continue to pay off, or to prevent unnecessarily continuing to search dead-ends when further progress seems implausible. Here, the run will continue as long as improvements continue to be made. A maximum number of unproductive generations is used to terminate a run.

Examination of different measures for stagnation and premature convergence could be most promising together with newly invented methodologies to abruptly terminate a run and commence a new run with the knowledge gained from the previous run. Moreover, a thorough study on a dynamic termination method that is based on the combined structural and behavioural diversity is suggested.

5.5 Innovative Ideas

The improvements detailed in this section contain some pioneering modifications to the canonical GP algorithm to enhance its performance.

5.5.1 Simplification

An approach to online simplification was introduced in [339], where programs were automatically simplified during the evolution using algebraic simplification rules, algebraic equivalence and prime techniques. The proposed method was tested on the regression and classification problems, showing its superior performance when compared with the standard GP systems.

5.5.2 Modularisation

The technique of automatic function definition (ADF) was introduced by [173] to potentially define useful functions dynamically during a run and to accelerate the discovery of solutions in GP. The number of fitness evaluations that must be executed [174] can be considered as a reasonable measure of computational burden. In [174], ADF was shown to allow the discovery and exploitation of regularities, symmetries, similarities and modularity of the problem environment. It was shown that for simpler versions of problems ADF was not effective but as the problems were scaled up increasing benefits became evident.

Evolving modular programs was also investigated in [10] [12], where special mutation operators (compress and expand) defined modules from the developing programs at random, allowing modular programs to emerge using the Genetic Library Builder (GLiB). An approach for reusability was proposed in [120] based on ADF, incorporating a library for keeping subroutines acquired by ADF. This library preserved knowledge and ensured reusability so that the acquired subroutines could be shared and reused. The most frequent sub-trees, which were expected to contain useful partial solutions, were grouped as modules [268]. Such sub-trees were encapsulated by representing them as atoms in the terminal set. Additionally, a random sub-tree selection and encapsulation was examined and empirical results illustrated performance improvement over standard GP. A method for automatically generating useful subroutines by systematically considering all small trees was presented in [43]. This algorithm moved progressively and systematically through the best trees of a given size and considered them as candidates for subroutine generation. This algorithm was successfully tested on the artificial ant problem.

Layered learning has been used for solving GP problems in a hierarchical fashion. The layered learning approach [57][125][126][141][295] decomposes a problem into subtasks, each of which is then associated with a layer in the problem-solving process. It is believed that the learning achieved at lower layers when solving the simpler tasks directly facilitates the learning required in higher subtask layers. Two program architectures are proposed in [142] for enabling the hierarchical decomposition based on the division of test input cases into subsets, each dealt with by an independently evolved code segment. The main program branch includes calls to these new entities via an expanded terminal set. The proposed technique offered substantial performance improvements over the more established methods such as the ADF for the even-10 parity problem.

A sub-tree was randomly selected in Module Acquisition (MA) [11][161] from an individual and then a part of this sub-tree was extracted as a module and preserved in a library defined as a new function. This module was protected against blind crossover operations and could be referred to by other individuals. In Adaptive Representation GP (AR-GP) [271], an effective sub-tree was selected and added as a new function to the function set to improve learning efficiency.

For GP to be able to address more demanding larger and more complex solution programs, it is inevitable for GP to have the ability to scale up. One way to achieve this is through more efficient modularisation practices. Further theoretical and empirical studies that aid in understanding the concept of building blocks within GP, their early detection and their further development and enrichment would certainly be a promising way to explore new schemes or possible improvements of current methods. In addition, a study that combines the current approaches may provide new insights in this area.

5.5.3 Other Innovative Ideas

Double-based Genetic Algorithm (DGA), which improves the performance of a GA, was shown to be relevant for GP paradigm [45]. Two types of doubles were defined based on permutations on the arguments and permutations on the terminals of the terminal set, introducing doubles in the population set. The Double-based Genetic Programming paradigm provided a useful extension of the GP standard search procedure and demonstrated its advantages for Genetic Programming.

The Best SubTree Genetic Programming (BSTGP) [234] selects the best sub-tree in order to provide the solution of the problem. This is different from the canonical GP, where the fitness of a tree is given by its root node. BSTGP also produces smaller trees as nodes that do not belong to the best sub-tree are deleted. The proposed approach was tested using a number of symbolic regression and classification problems showing comparable results to standard GP.

An approach using a clustering method was described [17] to reorganise subpopulations in GP, with the goal of producing more highly fit individuals. The initial population P is divided into number of subpopulations S_i after a nominated clustering frequency and according to the genetic similarity of the individuals. The sizes of the subpopulations are proportional to the average fitness of the individuals they contain. It was shown that a slight speedup over the canonical GP was observed for the multiplexer, parity and artificial ant problems.

The evaluation of a generation is widely accepted to be the most expensive process in GP. Sub-tree caching and vectorised evaluation [153] attempted to make this less expensive and more efficient. Two types of bottom-up and top-down caching were introduced, where the latter encouraged the caching of big sub-trees and the former encouraged the caching of small sub-trees. Although the caching of big sub-trees made the evaluation process more efficient, it was less likely that it could be matched and used again during the evaluation process due to its larger size.

It should be noted that all the other innovative ideas proposed by various researchers, which are specific to either the GP components and operators or the GP aspects and concepts that they pertain to, have been grouped, categorised and discussed in the other sections of this chapter.

5.6 Solutions to Known Issues or Problems within GP

The improvements detailed in this section endeavor to remedy an acknowledged issue within GP, such as the bloat phenomenon or lack of diversity etc.

5.6.1 Closure

The initial population-generating algorithms may not always generate valid individuals. The Grammar-guided genetic programming (GGGP) attempted to address this known closure problem. The reader is referred to Section 5.7.2.

5.6.2 Premature Convergence

Various studies have been conducted to address the issue of premature convergence [243] [326][342]. The issue of Premature Convergence is mainly addressed by making improvements on the components of GP. The reader is referred to Section 5.4.

5.6.3 Diversity

There has been much work which has focused on diagnosing or remedying the loss of diversity within the Evolutionary Computation. A new method of approximating the genetic similarity between two individuals was presented in [92], which used ancestry information to examine the issue of low population diversity. By defining a new diversity-preserving selection scheme, genetically dissimilar individuals were selected to undergo genetic operation. This provided the means to alter the perceived fitness of individuals. The study of how multi-population GP helps in maintaining phenotypic diversity was conducted in [305]. In [224], negative correlation was examined to improve diversity and prevent premature convergence. A study to evaluate the influence of the parallel (GP) in maintaining diversity in a population was conducted in [88].

A two-phase diversity control approach was proposed by [326] to prevent the common problem of the loss of diversity in GP. The loss of diversity was prevented in the early stage through a refined diversity control (RDC) method with Automatically Defined Functions (ADF) and a fully covered tournament selection (FCTS) method. RDC was an extension to general diversity control (GDC), which passed the diversity check if two whole program trees, with the main tree and ADFs treated as a whole, were not exactly identical in genotype. RDC treated the main tree and ADF as individual objects and hence both were required to be unique for the diversity check to pass. FCTS was an extension to the standard tournament selection (STS). It was argued in [326] that due to the randomness of STS, individuals with bad fitness may be selected multiple times, where an individual with good fitness may never be selected. FCTS avoided this issue by excluding individuals that had already been selected. The proposed methods effectively improved the GP's performance resulting in the reduction of number of generations needed to reach an optimal solution and decreased incidences of premature convergence.

5.6.4 Bloat and Code Growth

Many researchers have highlighted the problem of bloat, which is the uncontrolled growth of the average size of an individual in the population. There exist numerous studies of code bloat in GP [26][155][190][226][242][290][291][325]. Three principal approaches were summarised in [191] to prevent bloat, namely i) Limiting tree depth to some maximum value, ii) Using parsimony pressure, with the use of multi-objective (MO) methods and iii) Tailoring genetic operators such as size-fair crossover [192] [194] or fair mutation [193].

In the standard GP this issue is indirectly dealt with by limiting individual tree's maximal allowed depths. This can be viewed as unsatisfactory as this will require knowledge of the maximum necessary depth in advance of solving the problem. The effects and biases of size and depth limits on variable length linear structures were explored using empirical and theoretical analyses in [228]. It was argued in [342] that the increasing size of trees would reduce the speed of convergence towards a solution and thus affect the fitness of the best solution. Consequently, the dynamic maximum tree depth was proposed to avoid the typical undesirable growth of program size. A technique was demonstrated in [232], which significantly constrained the growth of solutions, i.e. bloat. This method imposed a maximum size on the created individuals within the population, which solely depended on the size of the best individual of the population. It was shown that the combination of depth limiting and methods which punish individuals based on excess size, were effective [213].

One mechanism for limiting code size is the Constant Parsimony Pressure, where larger programs are penalised by adding a size dependent term to their fitness [289]. This technique incorporated the program size as an additional constraint, but a hidden objective. The application of parsimony pressure was investigated in [95] in order to reduce the complexity of the solutions. Their results reported that while the accuracy on the test sets were preserved for binary classification setup, the mean tree size was significantly reduced. Parsimony pressure has also been used in [210] [211] to fight Bloat. Parsimony pressure incorporated in a multi-objective framework has been used by many researchers [270][340][341]. The use of multi-objective optimisation for size control was studied in [59]. Multi-objective techniques in the context of GP were investigated in [27] to reduce the effects caused by bloating. The inclusion of the tree size measure as one of the objectives has been found to be extremely effective at controlling bloat. It was shown in [20] that mutation can be used to prevent population collapse, a phenomenon where the population in Multi-objective GP rapidly degenerates to just trees of a single node because it tends to produce a positive mean increase in tree size per generation counterbalancing the parsimony pressure exerted by the fitness-based selection process. In [148], the functionality was first optimised and then afterwards followed by size (Ranking Method-Two Stage). The advantage of this was that pressure on size would not deter GP from discovering good solutions. This was because pressure is only applied when the individual has already reached the desired performance. However, bloating continued in solutions that had not attained the aspired performance.

The genetic operators could be modified to address the problem of bloating such as Deleting Crossover in [29]. In [70], speed improvements were observed by removing introns. In [255], the issue of too long solutions and bloat were addressed by Maximum Homologous Crossover (MHC). Equivalent structures from parents were preserved by aligning them according to their homology. MHC was tested on a symbolic regression problem and demonstrated its abilities in bloat reduction without inducing any specific biases in the distribution of sizes, allowing efficient size control during evolution. It was evident from the results that control of the size was possible while improving performance. The use of multiple crossovers was explored as a natural means to contain code growth [294]. Multiple crossovers were performed between two parent trees, where the total number of crossovers occurring between the two selected parents is dependent on the sum of the sizes of the parents involved. Three similar multi-crossover algorithms were shown to be a viable choice for containment of code growth.

It was argued in [324] that a significant problem with GP was the continuous growth of individual's size without a corresponding increase in fitness. A self-improvement operator (SI) was applied in combination with a characteristic based selection strategy to reduce the effects of code growth. Instead of simply editing out non-functional code the proposed method selected sub-trees with better fitness. The SI operator selects individuals that have at least one sub-tree that has a higher fitness value than its original tree (mother individual).

This will result in a fitter individual to replace a less performing individual with the reduction in depth and node count from its original tree. In other words, the sub-tree is upgraded to a new individual by removing the branches of the original mother tree. The performance of the proposed method was validated by testing it on a symbolic regression and a multiplexer problem showing a substantial reduction of code growth while maintaining the same level of fitness. It may be argued that the proposed approach is suitable for the above problems because they are simply and decomposable.

It was claimed in [336] that code bloat, slowed down the search process, destroying program structures, and exhausting computer resources. Non-neutral offspring (NNO) operators and non-larger neutral offspring (NLNO) operators were proposed to deal with these issues. An offspring could be considered as improved, neutral or worsened with respect to their fitness in comparison with their parents. The neutral offspring that are larger in size than their parents (LNO) could be further separated from those that are smaller in size than their parents (SNO). An LNO has more introns while an SNO has less. But they both have the same exon structure as their parents. It is argued that evolutionary processes favor LNOs, resulting in intron growth with no performance improvement. The proposed approach discarded all neutral offspring, named non-neutral offspring operators (NNO). Another approach, called non-larger neutral offspring (NLNO), kept SNOs and discarded only LNOs. Both approaches confined intron growth to different degrees. These two kinds of neutral offspring controlling operators were tested on two GP benchmark problems, namely symbolic regression and multiplexer problems to verify whether they could successfully apply parsimony pressure. It was concluded that NLNO was only able to confine code bloat and simultaneously improve performance.

It was shown in [296] that by eliminating bloat the performance of GP could be improved. Some modifications to the selection procedures were presented to eliminate bloat without deteriorating performance. The relationships of the bloat phenomenon with parallel and distributed GP has been investigated by many researchers and positive results have been obtained, where the bloat phenomenon could be controlled by parallelizing GP [76] [96]. It was shown in [63] that the parallel evolutionary model, specifically the island model, helped to prevent the bloat phenomenon.

A simple theoretically-motivated method for controlling bloat was introduced in [258], which was based on the idea of dynamically and strategically creating fitness "holes" in the fitness landscape repelling the population. These holes were created by zeroing the fitness of a certain proportion of above average length offspring. This meant that only a fixed proportion of offspring, those which violated the length constraints, were randomly penalised.

Three methods for bloat control were presented in [252], Biased Multi-Objective Parsimony Pressure (BMOPP), the Waiting Room, and Death by Size. BMOPP was a variation on the pareto-optimisation theme which combined lexicographic ordering, pareto dominance and a proportional tournament. The latter two methods do not consider parsimony as a part of the selection process, but instead penalise for parsimony at other stages in the evolutionary process. In the Waiting Room, newly created individuals were only permitted to enter the population after having sat in the “waiting room” or queue for period of time proportional to their size. This provided smaller individuals a greater opportunity to spread. Death by Size chose individuals to die and be replaced based on their size.

Parsimony pressure is traditionally used to reduce the complexity of solutions. In [297] however, a negative parsimony pressure was applied for a financial portfolio optimisation problem in GP, preferring complex solutions rather than simpler ones. Negative parsimony pressure presumed that the principle of Occam's Razor inhibits evolution [297].

Favourable results were shown where in some instances it was better to apply negative parsimony pressure.

Recent studies support the hypothesis that introns emerge predominantly in response to the destructive effects of the variation operators. Although, it may be argued that introns can be considered as useful because they protect good building blocks, nevertheless it can at the same time run the entire population into stagnation due to bloat, which is the explosive and exponential growth of introns. In this instance, no feasible improvements can be observed as the population is merely exchanging introns during recombination. A potential future direction within this area would be the formulation of more efficient variation operators or methodologies that can reduce the destructive nature of existing operators.

5.7 GP Variants and Hybrids

It should be noted that this work is focused on the canonical GP and does not delve into the other variants and hybrids. A very brief summary and introduction to GP variants and hybrids is included (this section) to introduce the reader to these new GP variants in the hope that it may lead to insights or clues for possible ideas that may guide future improvements in canonical GP.

Variants of GP are differentiated by their different structures [112] [170]. There are many different genetic programming (GP) structures such as tree, linear and graph structures with many other forms of representations being investigated and continuously emerging. For example, in [149] a new kind of GP structure called linear-tree-structure together with its own crossover and mutation operations was introduced. A novel Genetic Parallel Programming (GPP) paradigm, with a Linear Genetic Programming representation, was introduced in [201] for evolving parallel programs. It was observed that parallel programs were more evolvable than sequential programs. In [40] considerable speed up in evolution was observed using the GPP paradigm, running on a Multi-Arithmetic-Logic-Unit (Multi-ALU) Processor (MAP) evolving parallel programs and then serializing them into a sequential program.

A low level modularisation strategy, called compressed GA (cGA), was presented for linear genetic programming based on a substring compression/substitution scheme. The purpose was to protect building blocks and foster genetic code reuse. There are many more other variants explored by various researchers such as the Gene Expression Programming (GEP), Gene Estimated Gene Expression Programming (GEGEP) an extension to GEP [68], Multi niche parallel GP [97], directed acyclic graphs (DAGS) [103], parallel Automatic Induction of Machine Code with Genetic Programming (parallel AIM-GP) [245], Genetic Network Programming (GNP) [115], Grammar Model-based Program Evolution (GMPE) [137] and many others [16][136] [207] [276]. There are also many various hybrids that have been researched, such as genetic programming neural network (GPNN) [265] [266], Ant Colony Programming [31], traceless genetic programming (TGP) [249]. Many researchers have looked into improving the newly proposed hybrid. For example, in [31] the problem of eliminating introns in Ant Colony Programming (paradigm based on Genetic Programming and Ant Colony System) was investigated. Some of the GP Variants are shown in Figure 5.6.

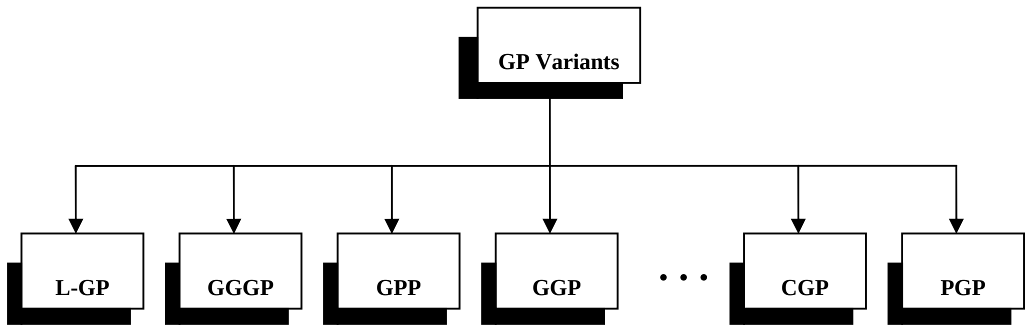


Figure 5.6 Some of the GP variants

5.7.1 Linear Genetic Programming

Linear GP makes use of linear phenome, which is a chain of instructions executed from left to right or top to bottom.

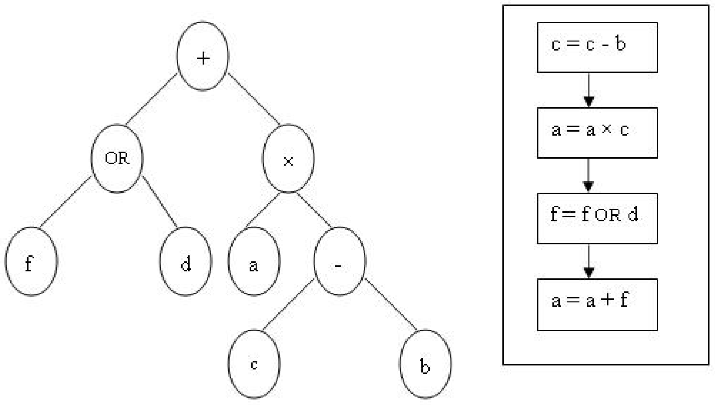


Figure 5.7 A tree structure phenome versus a linear phenome

The linear program and the tree program in Figure 5.7 are identical. In this figure, the first instruction is $c = c - b$. The effect of this instruction is to subtract the value in register b from register c and to place the result in register c. There are several linear GP systems, all sharing the same characteristic of the tree representation being replaced by a linear chromosome. Linear GP resembles conventional GA with the exception of the chromosome length being allowed to evolve, resulting in individual programs of different sizes. Each chromosome is a list of program instructions executed in sequence. Linear GP systems are typically divided into three groups: stack based, register based and machine code. Each program instruction takes its arguments from a stack in the stack-based GP [253], and then performs its calculation pushing its result back onto the stack. Machine code and register based genetic programming are comparable in that instructions are read and written from/to registers (see Figure 5.8). Their main difference is that in the machine code GP the instructions are real hardware machine instructions whereas in the register-based GP, programs are interpreted or compiled before execution.

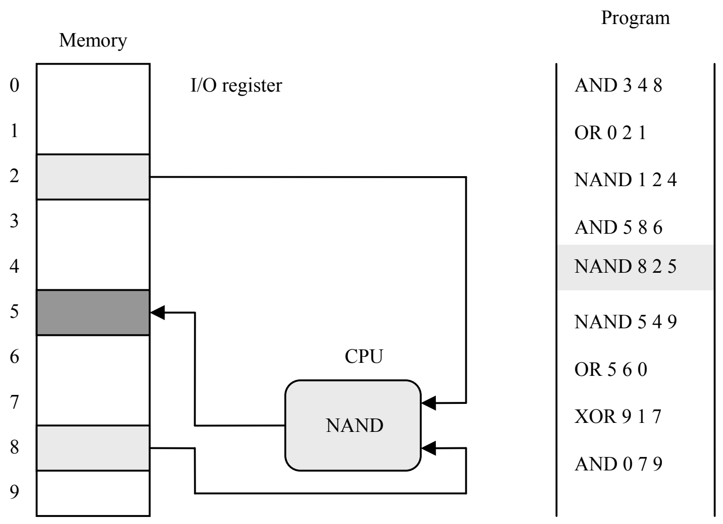


Figure 5.8 A sample program in register based GP

5.7.2 Grammar-Guided Genetic Programming

Grammar-guided genetic programming (GGGP) is an extension to standard GP with the aim to address the closure problem and simplify the search space [184][319] [321]. GGGP employs a context-free grammar (CFG) establishing a formal definition of the syntactical restrictions. Individuals are derivation trees that represent solutions belonging to the language defined by the context-free grammar [315]. GGGP always generates valid individuals (points or possible solutions that belong to the search space). In [316] the influence of program grammars on the efficiency of GP was described. In [335] a new method of representing based on CFG was used to separate search space from solution space through a genotype to phenotype mapping and this technique was applied to a symbolic regression problem showing improvement over a basic GP without a grammar. In [346] extensions to the operators were made to improve grammar-based evolutionary algorithms.

Many other variants of GGGP have also been investigated. For example in [116] [117], a new grammar guided genetic programming system called tree-adjoining grammar guided genetic programming (TAG3P+) was proposed. It is argued in [236] that standard GP is unable to search for all tree shapes, namely solutions that require very full or narrow trees. A different tree-based representation was used by [236] together with new local structural modification operators (point insertion and deletion) to eliminate this problem. The new representation was based on tree adjoining grammars (TAGs), which were first proposed in [145], to remove the fixed-arity limitation of standard GP.

5.7.2.1 Initialisation

A new initialisation method was introduced for GGGP in [41]. Random Branch tree-generation algorithm [41] guaranteed the generation of trees of requested size. However, this algorithm could not produce a well distributed set of trees, thus resulting with a negative impact on the convergence speed [105]. The Uniform Tree Generation algorithm [30] guaranteed the uniform creation of trees of requested tree size but was known to be too complex [98]. The Grow algorithm was modified in PTC1 and PTC2 to ensure that the trees were generated around an expected size [209].

A new tree-generation algorithm for GPPP, grammar-based initialisation method (GBIM), was proposed in [98]. A parameter was included to control the maximum size of the trees to be generated and thereby the initial populations generated were distributed in terms of tree size. It was shown that the proposed method had a higher convergence speed when compared with Ramped Half-and-Half, Basic, Random Branch, Uniform and PTC2 tree generation algorithms for an arithmetical equalities problem and the real-world task of breast cancer prognosis.

5.7.2.2 Variation Operators

The strong context preservative crossover operator (SCPC) was proposed in [50] to preserve the context in which the sub-trees occur in the parent trees and control the code bloat. Nodes with matching coordinates (see Figure 5.9) can be selected as crossover points in SCPC. In this figure for example, (2,1,3,1,2) would be the node by taking the 2nd branch counting left to right at the root of the tree, then take the first branch at that level, then the third branch there, then the first and finally the second. A node's position can therefore be described by a tuple of *n* coordinates $T = (b_1, b_2, ..., b_n)$, where *n* is the depth of the node in the tree, and b_i indicates which branch to choose at level *i*. In other words, the crossover is restricted to nodes that reside in similar contexts within the individual. In this case, emphasis was placed on the genotype of the individual (locations of the nodes), not the phenotype.

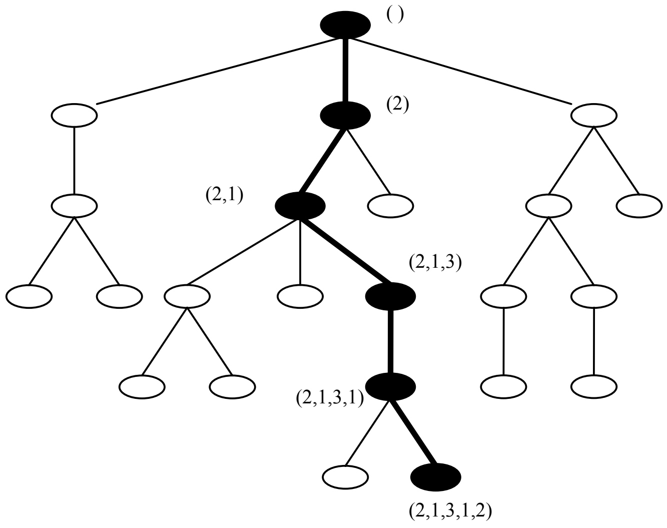


Figure 5.9 Tree coordinates

Crossover in GP has been blind [112] in contrast to biological crossover, where chromosomes exist with a matching and aligned homologous partner, in a process referred to as meiosis. The Fair crossover operator [48] was designed to prevent code bloat, which is a modified version of the operator proposed by Langdon [192]. Two original genetic operators, crossover and mutation, were proposed in [47] for the grammar-guided genetic programming (GGGP) paradigm. The grammar-based crossover operator (GBC) improved the GGGP performance, by providing a good balance between search space exploration and exploitation. The grammar-based mutation (GBM) operator generated individuals that matched the syntactical constraints of the CFG that defined the programs. The proposed operators were tested in two experiments demonstrating a higher convergence speed and a lesser likelihood of being trapped in local optima.

A new grammar-based crossover (GBX) operator was introduced in [217] for the grammar-guided genetic programming system to prevent code bloat. Moreover, GBX provided trade-off between exploration and exploitation of the search space. Grammatical Evolution (GE) is an extension of GP and evolves complete programs by using a Backus Naur Form (BNF) grammar or style of notation. Various different crossover operators were proposed in [106] [107] and in [108] a meta-grammar was introduced into GE allowing the grammar to dynamically define functions.

5.7.3 Parallel Genetic Programming

The measurement and computation of population fitness consumes a large amount of computational effort and is generally considered time-consuming. Many researchers have looked into distributing the computational effort needed to calculate fitness, hence Parallelizing GP which is frequently the focus of the parallel computing community. In addition, many researchers have investigated the spatial distribution of GP models. The two fields of parallel GP and spatially-distributed GP models had different goals. The main goal for parallel GP was often to speed up computation of the fitness evaluations, by either having each individual or some subpopulation evaluated on a separate processor. In this instance, the population as a whole was often treated as a panmictic population. The spatially-distributed GP constructed some form of spatial structure with the intention of maintaining diversity, which could be executed on the same processor or multiple processors.

Two basic approaches to parallelisation were discussed in [170]. In Distributed GP the population is divided into sub-populations (island model), each assigned to a processor (Figure 5.10).

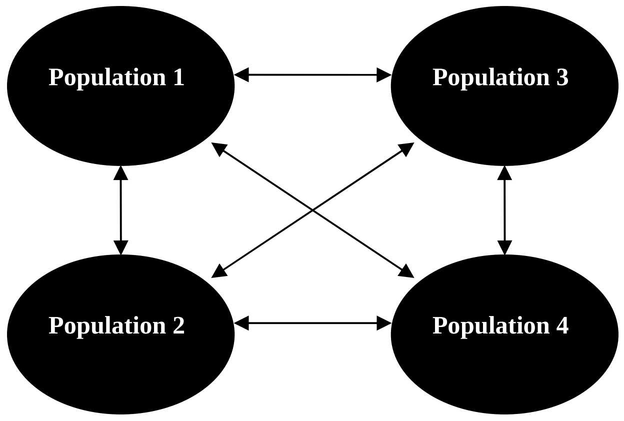


Figure 5.10 Distributed GP with 4 islands

The Distributed GP can be implemented on a network of workstations or a parallel computer where the GP operates on each sub-population separately. A specified percentage of individuals within each sub-population are selected for migration after a certain designated number of generations. There are many variations of distributed models such as demes, islands and niching methods. Figure 5.11 Shows a ring and a lattice stepping stone model.

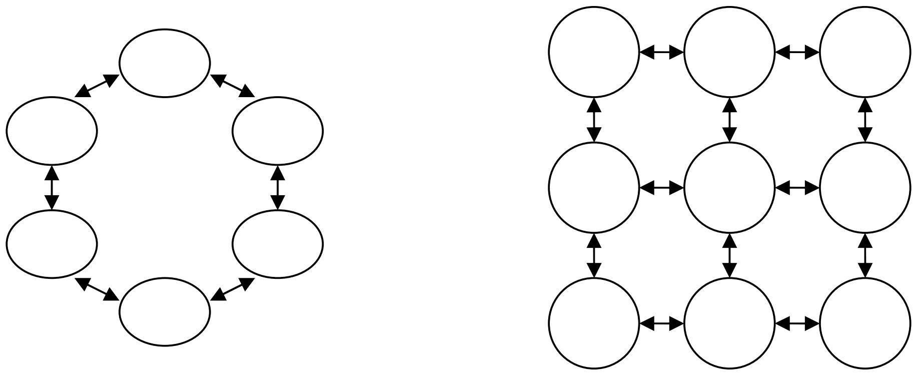


Figure 5.11 Stepping Stone Model as ring and lattice

For example, in the island method a population *P* of *M* individuals is divided into *N* subpopulations (called demes) $D_1, \ldots, D_N$ of *M*/*N* individuals. A standard GP works on each deme and the subpopulations are interconnected according to various communication topologies and information is periodically exchanged by migrating individuals from one subpopulation to another. As a result various new parameters such as the number of subpopulations *N*, number of individuals to be migrated (migration rate), the number of generations after which migration should occur (frequency) and the migration topology are needed in this methodology.

The first to analyze the behavior of distributed GP with respect to sequential GP was Punch [262]. In the second approach, there are no sub-populations or migrations, where steps are executed locally and asynchronously on a distributed basis. The independent algorithm tasks are distributing to separate processors.

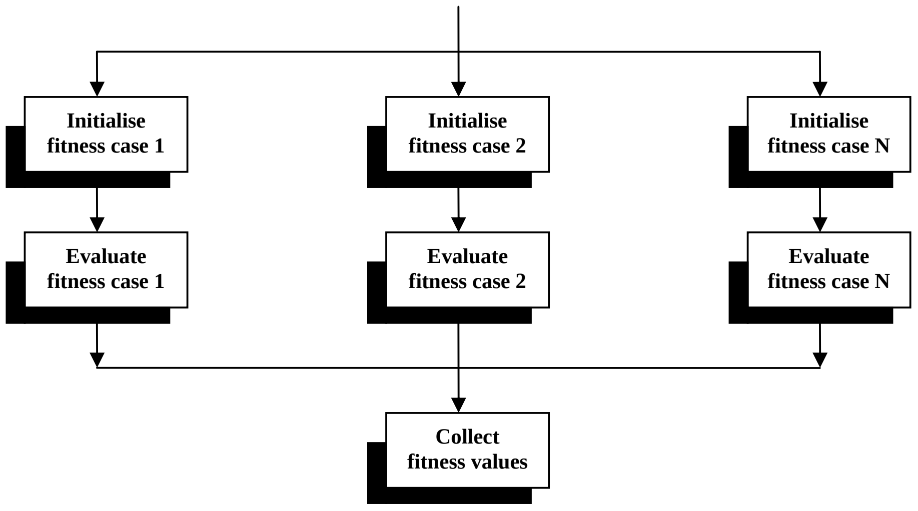


Figure 5.12 Parallelisation at the level of fitness cases [170]

Koza described three levels of parallelisation for the determination of fitness, at fitness cases (Figure 5.12), at fitness evaluation for different individuals and at independent runs (Figure 5.13).

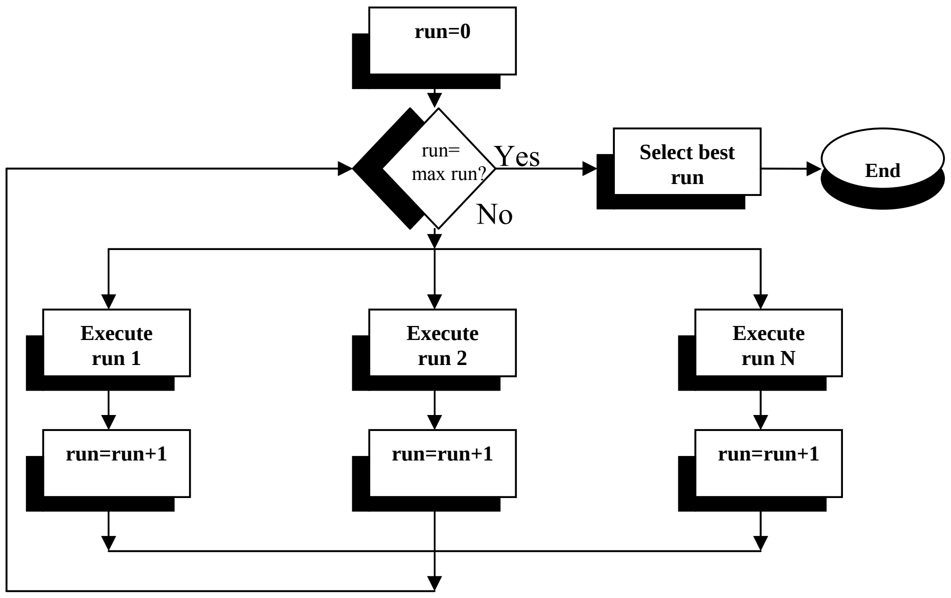


Figure 5.13 Parallelisation at the level of independent runs [170]

In [7], each processor was responsible for the fitness evaluation and breeding of a sub-population increasing the efficiency of GP. It was hypothesised in [305] that distributed GP outperforms the panmictic GP due to maintaining diversity. In [182] it was argued that increases in computational power can be realised by parallelizing the application. The parallel implementation of GP on a network of processing nodes was described in [8] that achieved super-linear performance.

In [261] multi-populations were examined and in [76] [80] various control parameters for multi-population models were systematically studied. In [203] layered genetic programming (LAGEP) was proposed based on multi-population genetic programming (MGP). This method employed layer architecture to arrange multiple populations. A layer contains a number of populations. In addition, an adaptive mutation rate tuning method was proposed to increase the mutation rate. LAGEP achieved comparable results to single population GP in much less time. The GP was used with several isolated subpopulations, where the individuals among the several populations were not allowed to communicate [77]. This methodology was referred to as Isolated Multipopulation Genetic Programming (IMGP). It was shown that although IMGP was not always helpful in obtaining better results, in some instances better results were obtained than in the classic method. A divide and conquer strategy was introduced to increase the probability of success in GP [85], where the search space was partitioned in smaller regions that were explored independently of each other.

A fine-grained parallel implementation of GP through cellular model on distributed-memory parallel computers with good performances was presented in [88]. In the fine-grained (grid) model, also called cellular, each individual is associated with a spatial location on a low-dimensional grid, interacting only with their direct neighbours. Different neighbourhoods can be defined for the cells. Some examples of the two-dimensional (2-D) neighbourhoods are the 4-neighbor (von Neumann neighborhood) and 8-neighbor (Moore neighborhood).

Various researchers have also attempted to make improvements to the canonical parallel evolutionary algorithms, e.g. in [101] the Speciating Island Model (SIM) was explored. In [76] [77] the aspect of population size was investigated for the multi-population Parallel Genetic Programming. It was discovered that an optimal range of values exists to speed up the search for solutions. In [82], the plague operator was used to enhance the performance of parallel GP (based on the island model). Individuals were removed every generation, altering the population size. This compensated for the increase in size of individuals and hence saved computational effort. Changing population size was also investigated for distributed GP in [269] to reduce bloat. It was shown that by keeping their size as small as possible and the amount of resources needed was decreased. There have been many other approaches in parallelizing GP [67] [146] [240][250] [277][301]. An extensive survey on the subject can be found in [304].

5.7.4 Graph Genetic Programming

In Graph Genetic Programming (GGP) system the GP operates on graphs. In [238], the notion of graph isomorphism was discussed and it was empirically shown how using a canonical graph indexed database (fitness database) can improve the performance by reducing the number of fitness evaluations and thus saving considerable evaluation time.

5.7.5 Cartesian Genetic Programming

A new form of GP called Cartesian Genetic Programming (CGP) was introduced in [230] in which programs were represented as indexed graphs (rather than as parse trees), encoded in the form of a linear string of integers. In [323], the CGP programs were represented as directed acyclic graphs (DAGs), enabling outputs from previous computations to be reused. An implicit context representation for CGP was described in [36] showing the beneficial effects of recombination to outperform the conventional Cartesian GP. The computational efficiency of graph-based Cartesian Genetic Programming was described in [231]. The Cartesian genetic programming was extended by utilizing automatic module acquisition in [312].

5.7.6 Page-based Genetic Programming

Page-based GP [241] is a linearly structured GP (L-GP), where individuals take the form of a “linear” list of instructions. A Page-based linear genetic programming (GP) was proposed in [112] [113] where individuals were described in terms of a number of pages. It was shown that page-based linear GP evolves solutions better than the block-based linear GP [244].

5.7.7 Other Representations

The performance of GP was improved by using a data structure coded by binary decision diagrams (BDDs), reducing storage requirements and accelerating the fitness calculation [331]. BDDs are a compact representation of Boolean functions using directed acyclic graphs. The entire population was stored as a shared BDD and all genetic operations and fitness calculations were performed on the BDD. This technique is suitable for problems where only Boolean variables and functions are involved. BDD-based GP is not practical for problems where real variables, such as symbolic regression are used. Nevertheless, it can also be used for integer-based programs by encoding the integers as binary vectors. New crossover, mutation and evaluation algorithms were developed for BDD [331].

A technique to reduce the time and space requirements of GP was proposed in [103]. The population of parse trees was stored as a directed acyclic graph (DAG). However, it was stated that this technique cannot be applied to all problems due to restricted program encoding and bounded fitness cases, such as the Artificial Ant and Cart Centering problems. The number of nodes stored and evaluated was reduced by a significant factor resulting in less space requirements to store a population of computer programs. In addition, time savings were also observed as a result of caching. In the standard sub-tree crossover it is difficult to make changes near the root, occasionally causing runs to become trapped in local maxima. Based on these structural limitations a different tree representation, AppGP, was proposed [91]. The representation of trees and the tree manipulation algorithms were modified in AppGP. All non-terminal nodes were represented as application (APP) nodes and the AppGP representation had more nodes than the standard GP representation, providing more potential points for the application of recombination operators. It was shown that on all of the test problems, AppGP did no worse than standard GP, and in several instances it outperformed standard GP.

5.7.8 Memetic Algorithms - Hybrids

Researchers have attempted to capitalise upon the strategies of other methods by incorporating them into an enhanced version of GP that outperforms the canonical GP. The GA-P [123], which is a genetic algorithm and genetic programming hybrid, performed symbolic regression by combining the conventional GA function optimisation strength with the GP paradigm to evolve complex mathematical expressions. The GA-P was extended in [278]. A genetic algorithm (GA) was embedded into a genetic programming (GP), where each paradigm operated at different levels within the problem domain [37].

The genetic programming paradigm was hybridised with statistical analysis in [1] to derive systems of differential equations. A framework for combining GP and inductive logic programming (ILP) was proposed in [320]. A memetic algorithm was proposed in [35] evolving heterogeneous populations, in which a GA was used to optimise the numeric terminals of programs evolved using GP. The GP algorithm was hybridised with hill climbing and the nature of new crossover algorithms, crossover hill climbing (XOHC) and crossover with Simulated Annealing (XOSA), was investigated [246]. It was shown that the hybrids offer added search power and hybridizing GP with hill climbing yields better results than the standard GP.

CHAPTER 6

POPULATION VARIATION

"Ideas are like stars: you will not succeed in touching them with your hands, but like the seafaring man on the ocean desert of waters, you choose them as your guides, and following them, you reach your destiny.", Anatole France.

6.1 Preliminary Discussions

Genetic Programming, like other approaches in Evolutionary Computation, is normally computer processing intensive and therefore special methods are necessary in order to increase or enhance its performance. The main drawback of GP has been this often large amount of computational effort required to solve complex problems. Researchers have investigated various methods for overcoming this problem. This issue of computational effort was addressed in [154] by introducing a number of different methods for improving GP execution time. While such investigations were mainly focused on the design and realisation of the GP platform from a software implementation point of view, many other researchers have considered techniques to improve efficiency from the GP paradigms perspective. A wide number of different techniques for improving the efficiency of GP have already been proposed. Some of these include the development of the Automatic Defined Function (ADF) methodology [173][174] used to improve the GP search efficiency and the brood recombination operator [299], which was introduced as a substitute for the Standard Genetic Programming (SGP) crossover. Other efficiency techniques have also been proposed such as partitioning the search space into smaller regions [85], parallel implementation of GP [8] and bloat control methods to reduce tree size[213]. The role of population size in GP was first but very briefly studied by Koza [170]. In his study, the 6-multiplexer problem was solved using various population sizes. The study concluded that a larger population size *M*, for generations between generation 0 and generation *i*, increased the cumulative probability *P*(*M*,*i*) of satisfying the success predicate of a problem for GP. The population size was maintained at a constant level and was not varied throughout the run. The plague operator was then introduced by Fernandez *et al.* [79] to fight bloat, which many argue is a phenomenon commonly observed when algorithms use variable size chromosomes (data structures). Plague was defined as an operator that suppressed the population by removing a fixed number of individuals from the population at each generation *g*. It was shown that a given fitness level could still be reached with a smaller computational effort by removing individuals at a linear rate. In [164], Kouchakpour *et al.* carried out a study on the effects of population variation conducted during the run of the GP paradigm and investigated whether the nature of the “population variation”, i.e. the way the population is varied, has any significant impact on GP performance. The proposed general population variation scheme is known as Population Variation (PV) and some of the variants of this scheme are investigated in this chapter. The performance of the plague operator is also discussed and compared with that of the proposed population variation scheme.

6.2 An introduction to Population Variation

In genetic programming a population or a set of solution points undergo adaptation simultaneously. It is believed that population control plays a significant role in the success of these algorithms [170]. Consequently, a generic technique called Population Variation (PV) is proposed with the main aim of reducing computational effort and to possibly accelerate convergence towards a solution, where convergence is defined as the GP approaching a solution with a desired fitness. The parameter λ is defined as the required Average number of Evaluations to a Solution (AES) before a successful termination. In addition, a small AES is defined to represent fast convergence and large AES slow convergence. The computational effort is measured as the total number of individuals evaluated in one run of the GP. In the classic GP (Standard GP, SGP), the population remains constant during the run and hence the Population-Generation profile $P(g)$, which is the total number of individuals (or the population size) at any generation g, is described graphically by a horizontal line with a y-intercept of M (population size), as depicted in Figure 6.1. The generation number "g" ranges from 0 (the initial generation containing the initial random population) to G, which is the maximum number of generations to be run.

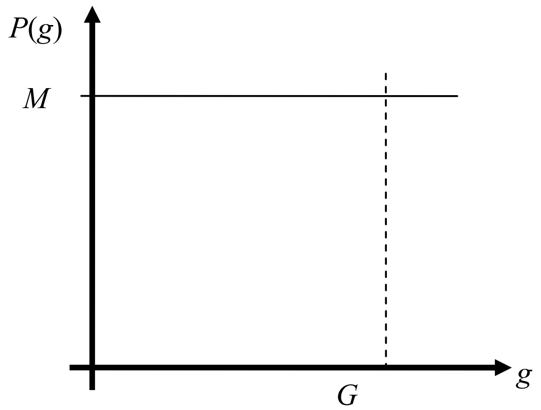


Figure 6.1 Population profile in SGP

Using our definition, the Computational Effort per run ε can be computed according to (6.1) as the sum of the Population-Generation profile over all generations, which is equivalent to the total number of evaluations, which can be approximated by using integration.

$$\varepsilon = \sum_{g=0}^{g=G} P(g) \approx \int_{0}^{G} P(g)\, dg \qquad (6.1)$$

In [79], plague was used to indirectly fight bloat, by reducing population size. The computational effort at a given generation *g* was defined as the total number of nodes evaluated from generation 1 to generation *G*. It is believed that this viewpoint of computational effort is at a microscopic level and is not necessary for the following two reasons. Firstly, given today's computer power, the nodes which are extraneous due to bloat have very little impact on the total computational effort. In Figure 6.2, two individuals are shown that produce the same fitness but one is longer in length due to bloat. It is believed that bloat should at worst case, if any, be considered a hindrance to finding a solution but not a burden on computational effort.

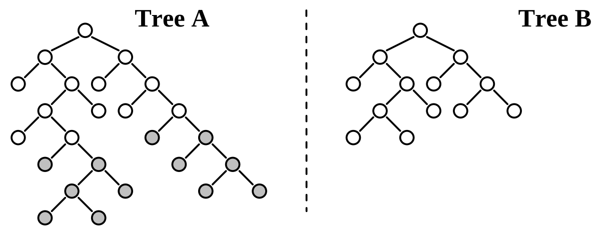


Figure 6.2 Two successful trees, Tree A contains introns

Secondly, it should be noted that the tree length, i.e. the number of nodes in a genotype, in general does not have to correlate with the computational effort required to evaluate the corresponding phenotype. For instance, consider a genetic program being used for driving a mobile robot, where language tokens are used for driving or turning a robot. The evaluation of such a driving token may be in the order of 1000 times more computationally expensive than some arithmetic token. No meaningful correlation between tree length and evaluation time exists if an iteration construct exists in the GP language. Consequently, a program's length does not allow any conclusion about its run-time. For the above two reasons, the calculation of computational effort is based on the number of individuals that are required to be evaluated in one run of the GP as defined by (6.1). The main goal of the PV scheme is to minimise computational effort such that the computational effort of the PV scheme ε_{PV} is better than or in the worst-case equal to the computational effort of SGP ε_{SGP}, expressed in the following manner.

$$\varepsilon_{PV} \leq \varepsilon_{SGP} \tag{6.2}$$

The worst-case upper limit per run for the computational effort of the Population Variation scheme $\varepsilon_{PV} = \sum_{g=0}^{g=G} P(g)_{PV} \approx \int_0^G P(g)_{PV}\, dg$ is set to be equal to the computational effort of a classical GP $\varepsilon_{SGP} = \sum_{g=0}^{g=G} P(g)_{SGP} \approx \int_0^G P(g)_{SGP}\, dg$, as defined by (6.3).

$$\int_0^G P(g)_{SGP}\, dg = \int_0^G P(g)_{PV}\, dg \tag{6.3}$$

This upper limit ensures that the computational effort for an unsuccessful run in the PV scheme will never exceed that of the SGP. It also ensures that at worst the PV scheme performs equally well as does a SGP, as far as computational effort to the end of an unsuccessful run is concerned. At the same time, it is hoped that the population variation scheme outperforms the SGP for a successful run, i.e. $\varepsilon_{PV} < \varepsilon_{SGP}$.

One of our innovations called Population Variation Linear Reduction (PV-LR), described in the next section, has a lot of similarities with the plague operator but at the same time has some major differences providing more benefits. The PV-LR is constrained to a worst-case computational effort scenario equal to that of the SGP per run. It begins with a much larger initial population than the SGP to increase its population diversity at birth as compared with the SGP. In contrast, the plague operator approach starts off with the same population as the SGP. The results, which were reported for the plague operator in [79], indicated an improvement over the SGP however the study used a very large initial population size for the SGP. To illustrate, the study used 5000 and 10000 individuals for both the artificial ant problem and the even parity problem. On the other hand, the study conducted in [170] used an initial population size of only 500 for the artificial ant problem and 4000 for the even parity problem. It is believed that this is not a valid comparison, as the plague operator should have been compared with a SGP at its optimal or at least close to optimal starting population size. There is a point after which the cost of a larger population, in terms of computational effort, begins to outweigh the benefit it introduces. This observation was also made in [170], that after a point the cost of a larger population (in terms of individuals to be processed) exceeded the benefit it obtained from the increase in the cumulative probability $P(M,i)$ of satisfying the success predicate. Consequently, the plague operator approach will also be investigated in this chapter and compared it with a SGP at its near-optimal population size.

6.3 Population Variation Variants

In this section some different variants of PV are discussed and investigated. These are Population Variation Reduction (PV-R), Population Variation Increment (PV-I) and Random Population Variation (PV-RAN).

6.3.1 Population Variation Reduction Scheme

The rationale behind the population reduction is that by increasing the initial population size to a much larger value than the initial population size of a SGP (i.e. $P_{PV}(0) >> P_{SGP}(0)$), the population diversity is increased. That should in turn aid in the acceleration of convergence towards a solution (in terms of number of generations produced) at the initial stages of the GP run. The reduction is required to progress in such a way that the computational effort at the end of the run is equal to that of a SGP.

The PV-R scheme deploys the following algorithm. Given the set of trees $T = \{T_1, T_2, T_3, \dots, T_M\}$, which form the M individuals in a population at generation g, sorted according to fitness $f(T_1) \leq f(T_2) \leq f(T_3) \leq \dots \leq f(T_M)$, where f is the fitness function, the k worst trees from a fitness point of view are removed at each generation. This elimination policy ensures that a good value of fitness is maintained across the populations since best performing trees are not suppressed. The Population Variation Linear Reduction (PV-LR), PV Step Reduction (PV-SR), PV Quadratic Reduction (PV-QR), and PV Exponential Reduction (PV-ER) belong to the family of PV-R schemes investigated in this chapter.

In PV-LR, the population is altered linearly by using the relationship defined as,

$$P(g) = m \times g + P_{PV}(0) \qquad (6.4)$$

where m ($m<0$) is the rate of linear suppression and can be derived using (6.3). The rate of linear suppression m, is defined by (6.5).

$$m = \frac{2 \times \left(P_{SGP}(0) - P_{PV}(0)\right)}{G} \tag{6.5}$$

In PV-ER, the population is altered using a negative exponential function approaching an asymptote of ρ as defined by (6.6).

$$P(g) = P_{PV}(0)e^{(-m \times g)} + \rho \tag{6.6}$$

Similarly, using (6.3) it can be shown that the rate of exponential suppression m is directly related to the asymptote ρ as shown below.

$$\frac{P_{PV}(0)}{\rho \times G}\left(1 - e^{(-m \times G)}\right) + \rho = P_{SGP}(0) \tag{6.7}$$

In PV-QR, the population is altered quadratically by using the relationship defined by (6.8) with the coefficients a and b defined by (6.9). The parameters defined for PV-SR are given in Appendix H. A sample population profile of these reduction schemes are shown in

Figure 6.3.

$$P(g) = -ax^2 + bx + P_{PV}(0) \qquad a, b \in \Re \vee a > 0 \tag{6.8}$$

$$6 \times P_{SGP}(0) = -2aG^2 + 3bG \tag{6.9}$$

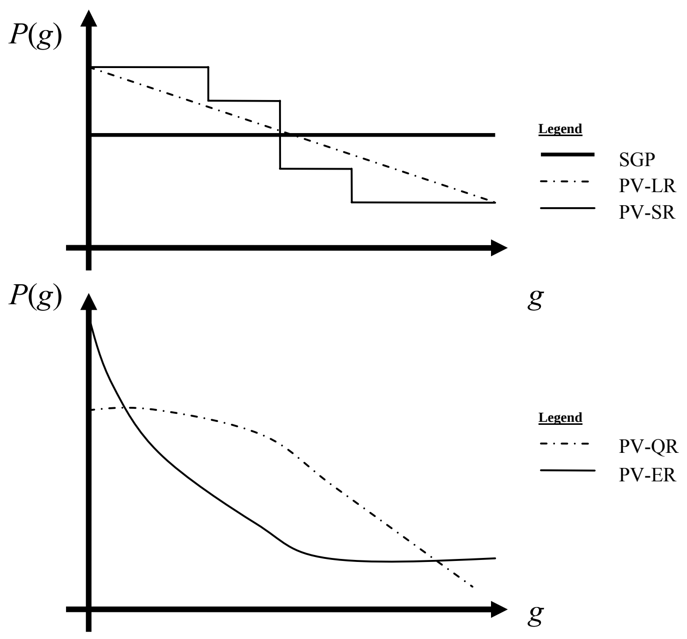


Figure 6.3 Population Profile-Population Variation Reduction

6.3.2 Population Variation Increment Scheme

In the PV-I scheme, we start off with an initial population size smaller than that of the SGP (i.e. $P_{PV}(0) << P_{SGP}(0)$), and increase the population as the generations progress. The PV-I scheme deploys the following algorithm for increasing population size. Given the set of trees $T = \{T_1, T_2, T_3, \ldots, T_M\}$, which form the M individuals in a population at generation g, sorted according to fitness $f(T_1) \leq f(T_2) \leq f(T_3) \leq \ldots \leq f(T_M)$, where f is the fitness function, the k best performing trees from a fitness point of view are selected and mutated at each generation. These mutated individuals are then added to the next generation. In this chapter one variant of PV-I is investigated, namely the Population Variation Linear Increment (PV-LI) as shown in Figure 6.4, satisfying (6.3).

The motivation behind this idea is that initially a large population size is not as effective as having a larger population size after the individuals have evolved and have a higher fitness value. It is believed that the increase in population and diversity at this time could play a more significant role than an increase in population size initially where the majority of the individuals are performing poorly. As generations proceed, the population size is incremented to a much larger value than the initial population size of a SGP (i.e. $P_{PV}(G) >> P_{SGP}(G)$). The increase is performed in such a way that the computational effort at the end of the run is equal to that of a SGP.

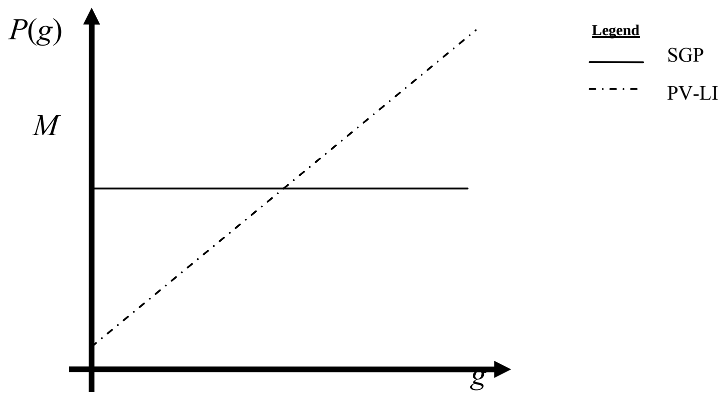


Figure 6.4 Population Profile SGP versus PV-LI

6.3.3 Random Population Variation

In the PV-RAN scheme, the population is randomly altered around a mean value ($\mu = P_{SGP}(g)$) within a minimum and maximum range ($P_{min} \leq P_{PV}(g) \leq P_{max}$), as shown in Figure 6.5. Due to the randomness of the population variation it is expected that (6.3) will hold. The incentive of this study is to see whether any kind of arbitrary population variation can result in performance improvements over SGP.

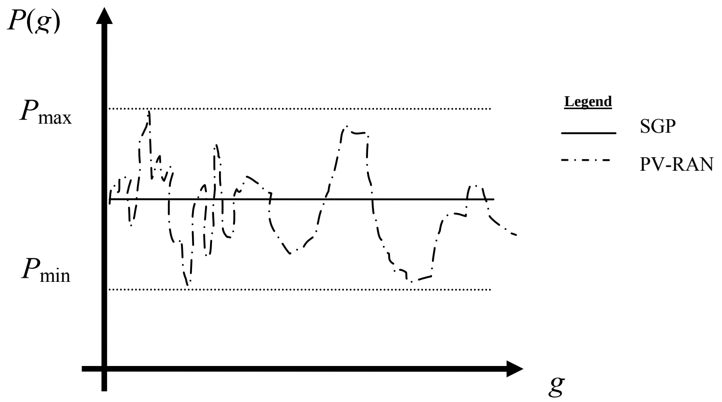


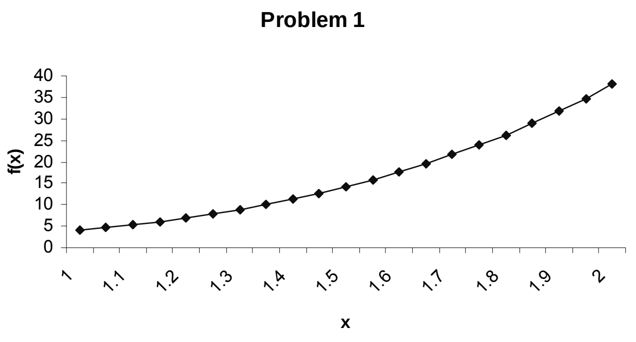
Figure 6.5 Population Profile SGP versus PV-RAN

6.4 Applications

The PV study within this chapter is based on four representative problems that have been classically used for testing GP, namely the symbolic regression problem, digital logic problems such as the parity problem and the Boolean symmetry problem, the sequence induction problem and the artificial ant problem. For each problem, the hypothesis is tested on two different landscapes. In most instances, the first landscape is more complex and the second landscape is a simplified variant of the first landscape. Following are brief descriptions of each problem.

6.4.1 Symbolic Regression Problems

In science and engineering, it is often required to determine the rule or function fit for a set of analytical data which is obtained from experimentation. Problems of identifying a mathematical expression that relates the independent variable to the dependant variable for a given finite sampling interval are defined as symbolic regression problems. In this study, the sample of data shown in Figure 6.6 is used.



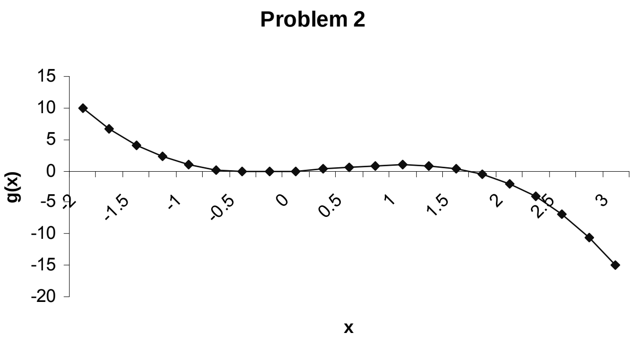


Figure 6.6 Sample data used as input for the regression problems 1 and 2

The first problem is best solved by a degree-4 polynomial as defined by (6.10) and the second problem is best solved by a cubic polynomial as defined by (6.11).

$$f(x) = 1.5x^4 + 1.5x^3 + x, \qquad x \in [1,2] \tag{6.10}$$

$$g(x) = -x^3 + x^2 + x, \qquad x \in [-2,3] \tag{6.11}$$

The inputs for both problems are composed of 20 equidistant points within the defined domains and the terminal set used for these problems is $\boldsymbol{T} = \{X\}$ and the function sets are given by $\boldsymbol{F} = \{+, \times, -, \%\}$, where % is the protected division operator defined by (6.12) to ensure the closure property. The symbols a and b are simply variables that can take real values. The defined terminal and function sets will satisfy the sufficiency property.

$$a\%b = \begin{cases} a/b, & \forall b \in \Re, b \neq 0 \\ 1, & b = 0 \end{cases} \tag{6.12}$$

Problem 1 is an example of symbolic regression with constant creation due to the presence of the coefficients in the target expression. As there are no ephemeral random constants in the terminal set, GP arbitrarily creates these constants. The fitness is defined as the sum of the absolute difference between the fitness case and the output of the GP.

6.4.2 Digital Logic Problems

6.4.2.1 Even Parity 4 Generator (Odd Parity 4 Checker)

One of the major concerns in data-handling systems, such as in digital transmission systems, is to avoid data transmission errors. The simplest approach to reducing the probability of error is to introduce the parity check, where the even parity *k* generator (Odd parity *k* checker) function $f(X_1, X_2, X_3, \ldots, X_k)$ with *k* Boolean arguments returns true (binary *1*) if an odd number of its Boolean arguments evaluates to true, otherwise it returns false (binary *0*). Herein the Even Parity 4 generator (Odd Parity 4 Checker) problem is used with $k=4$ and $2^4=16$ fitness cases. The fitness is computed as the number of hits over the fitness cases. Thus a perfect individual has a fitness of 16 and the worst individual has a fitness of zero (0). The terminal set is composed of four different Boolean variables $\boldsymbol{T} = \{X, Y, Z, W\}$ and the function set is given by $\boldsymbol{F} = \{NAND, NOR\}$.

6.4.2.2 Boolean 6-Symmetry Problem

The Boolean symmetry function is often used as a benchmark in the fields of neural networks and machine learning. The Boolean symmetry function $g(X_1, X_2, X_3, \ldots, X_k)$ with *k* Boolean arguments returns true if its Boolean arguments are symmetric. Symmetry is determined by verifying that the first argument matches the k^{th} argument, the second argument matches the k-1^{th} argument etc. Herein, the Boolean 6-Symmetry Problem is used with $k=6$ and $2^6=64$ fitness cases. Fitness is calculated as the sum over the fitness cases of the Hamming distance between the value returned by the program and the correct value of the symmetry function. The terminal set is composed of six different Boolean variables $\boldsymbol{T}$ =

{*X*,*Y*,*Z*,*W*,*K*,*P*} and the function set contains the *AND*, *OR* and Equivalence (exclusive-*NOR*) logic operations and is given by ***F*** = {*AND*,*OR*,*XNOR*}.

6.4.3 Sequence Induction Problems

Induction has been widely viewed as an important component of human intelligence. Sequence induction involves discovering a mathematical expression that generates any arbitrary element in an infinite sequence $T = \{T_0, T_1, T_2, T_3, \ldots\} = \{T_J\}$. Recursive and a simple sequence induction problems have been investigated as described below.

6.4.3.1 Recursive Sequence Induction

Recursive sequence induction is a sequence induction where the target function is not a straightforward function of the index position *J* of the sequence. The well-known monotonic divergent Fibonacci sequence is used in this problem, which is computed using the recursive expression $T_J = T_{J-1} + T_{J-2}$ with $T_0 = T_1 = 1$. The terminal set is composed of the index position *J* and four small integers 0, 1, 2 and 3, therefore, ***T*** = {*0*, *1*, *2*, *3*, *J*}. The function set is given by ***F*** ={+,×, -,*SRF*}, where *SRF* (Sequence Referencing Function) provides a facility to refer to previously computed indices less than *J*. To illustrate, the sub-expression $SRF(n,d)$ at index position *J* returns the value of the previously computed *n*th term, provided $0 < n \le J-1$ and otherwise it returns the default of *d*. It should be noted that the Sequence Referencing Function returns the value computed by the current symbolic expression not the actual correct value of the Fibonacci sequence. Fitness cases for this problem consist of the first 20 elements of the Fibonacci sequence. The fitness is computed as the sum of the absolute difference between the sequence value produced by the symbolic expression and the actual Fibonacci sequence.

6.4.3.2 Simple Sequence Induction

A simple sequence defined as *T*={*6*,*11*,*18*,*27*,*38*,…,*402*,*443*,…} is used for the second problem. The terminal set is defined as ***T*** = {*0*, *1*, *2*, *J*} and the function set is given by $\boldsymbol{F} = \{+, \times, -\}$. Fitness cases for this problem consist of the first 20 elements of the given sequence.

6.4.4 Artificial Ant Problem

Jefferson *et al.* [143] first devised a planning task involving an "artificial ant" attempting to traverse an irregular trail and successfully used the conventional string-based genetic algorithm to discover a finite state automaton enabling the "artificial ant" to traverse the trail. This involves the task of navigating an artificial ant to find all the food lying along an irregular trail. The first landscape used is the “Santa Fe trail”, which is an irregular winding trail consisting of 89 food pellets on a 32×32 two-dimensional toroidal grid in the plane, with the ant located in the top left corner facing east.

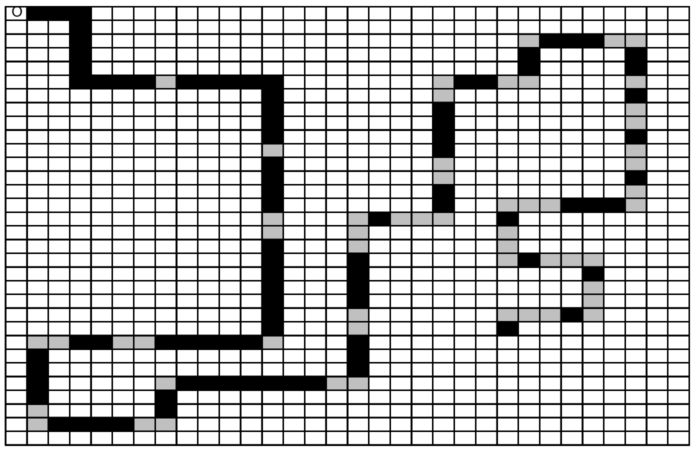


Figure 6.7 Santa Fe trail for the artificial ant

This trail is neither straight nor continuous, but contains single and double gaps and single, double and triple gaps at corners as shown in Figure 6.7. The gaps are indicated with a grey shade. The ant, indicated by a circle, is located in the top left corner facing east. The second landscape, which is considered a scaled-up version of the Santa Fe Trail, is the “Los Altos Hills trail”. It consists of 157 food pellets on a 100×100 two-dimensional toroidal grid in the plane with varying discontinuities in the sequence of food pellets. Figure 6.8 shows part of the Los Altos Hills trail (upper left 50×70 grid).

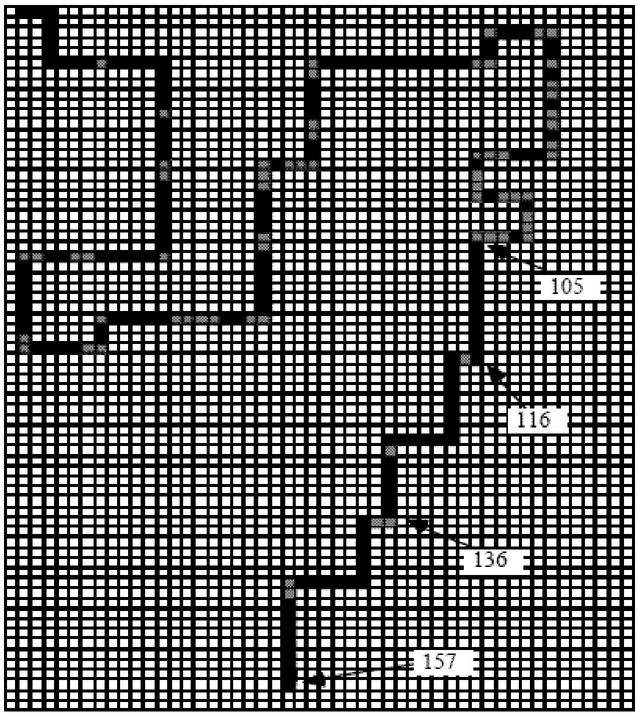


Figure 6.8 Los Altos Hills trail for the artificial ant

The artificial ant has a very limited view of its world. In particular, it has a sensor that can only see the single immediately adjacent cell in the direction it is currently facing. Here, the ant's goal is to traverse the entire trail and eat all the food within a reasonable amount of time. The function set consists of the *IF-FOOD-AHEAD* conditional branching operator and the two argument connective function for the Santa Fe Trail. The ant has been limited to 400 time steps in the first landscape. Hence, ***F*** ={*IF-FOOD-AHEAD*,*PROGN*}. The terminal set consists of the primitive functions to change the state of the ant, namely ***T*** = {(*MOVE*), (*TURN-RIGHT*), (*TURN-LEFT*)}. In the second problem the ant has been limited to 3000 time steps due to trial's more complex nature. The *PROGN3* and the *PROGN4* were added to the basic function set in the second problem. Therefore, ***F*** ={*IF-FOOD-AHEAD*,*PROGN*,*PROGN3*,*PROGN4*} and ***T*** = {(*MOVE*), (*TURN-RIGHT*), (*TURN-LEFT*)}.

6.5 Experimental Results

The following section summarises the experimental results obtained for the various applications using the theoretical concepts outlined in the previous sections. It is divided into four main subsections, namely symbolic regression, the digital logic problem, the sequence induction problem and the artificial ant problem. Each subsection is further subdivided to address the two different landscapes studied. Each subsection captures the results obtained for the four case studies. In the following, three different measures are used, namely percentage average total computational effort with respect to SGP, or computational effort ratio R_ε as defined by (6.13).

$$R_\varepsilon = \frac{\varepsilon_{PV}}{\varepsilon_{SGP}} \tag{6.13}$$

Here ε is the computational effort as defined by (6.1) for each respective algorithm. Secondly, the percentage AES with respect to SGP, or AES ratio R_λ is defined (6.14).

$$R_\lambda = \frac{\lambda_{PV}}{\lambda_{SGP}} \tag{6.14}$$

Here λ is the Average number of Evaluations to a solution (AES) required before a successful termination. The success rate SR ratio of the PV scheme in comparison with SGP is defined by (6.15) where SR is defined as the percentage of runs terminating with success.

$$R_{SR} = \frac{SR_{PV}}{SR_{SGP}} \tag{6.15}$$

The genetic programming paradigm was implemented as a tree-based GP using the C programming language and the experiments were executed on an Intel Pentium 4 processor machine. Twenty runs were required for each experiment with the total number of generations G used in the experiments set to 90, with the probability of crossover of 0.9 and probability of reproduction of 0.1. The results in the following subsections are the average of all runs, including the unsuccessful runs, for each problem and PV scheme. The initial creation of the population was ramped half-and-half with fitness proportionate selection and the elitist strategy. The timing for each combination of problem and algorithm varied. It took approximately 20 minutes for the easier problems to arrive at a solution, while the harder problems such as the Los Altos Hills trail took significantly longer of more than 20 hours to converge to a solution. It should however be noted that this is not a reliable measure for comparing different algorithms because it can vary depending on different PC CPU (processor) features and programming efficiency and style etc. The computational measure used in this study is a more appropriate measure, as it is specific to the GP algorithm itself and is independent of programming style or hardware etc, which are not related to the GP algorithm. It should be noted that the main aim is the reduction in total computational effort. In addition to lower computational effort, a faster convergence towards a solution and a higher success rate is a bonus and is considered to be a favourable result. An algorithm is considered superior if it can reach a specified fitness with less computational effort.

6.5.1 Symbolic Regression

6.5.1.1 Symbolic Regression with Constant Creation (degree-4)

The total computational effort for the degree-4 polynomial regression problem in comparison with the SGP, R_ε, is displayed graphically in Figure 6.9. The efficiency of the algorithm is measured by the amount of computational effort, i.e. the lower the computational effort, the more efficient the algorithm. From Figure 6.9, it can be observed that all the population variations, including the plague operator, outperform the SGP. Therefore, our results agree with Fernandez *et al.* [79] for the amount of computational effort saved when using plague over SGP. In addition, it can be seen that the PV scheme proposed in this study achieved better results than the plague operator for this problem.

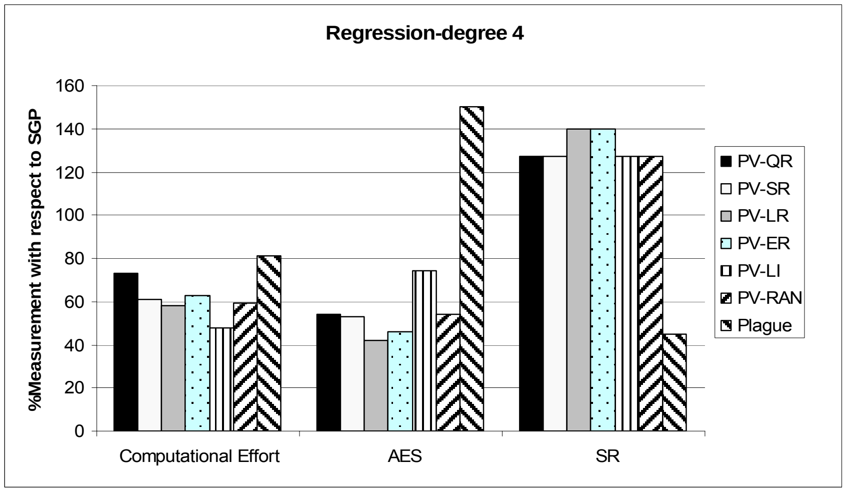


Figure 6.9 Performance Measures with respect to SGP for the degree-4 regression problem

The amount of reduction in population amongst the variants is in the following order: PV-QR, PV-SR, PV-LR and PV-ER. It can be seen that as the amount of reduction is increased, there is an improvement in terms of computational effort. However, the improvement stops once the amount of reduction is beyond a certain threshold. Any reduction beyond this value can be considered to be detrimental as the GP does not have a sample space big enough to perform effectively. This can clearly be seen in Figure 6.9 as PV-LR is better than PV-SR but PV-ER performs worse than PV-LR. Therefore, the optimal reduction is somewhere in the vicinity of the linear reduction scheme. Consequently, there will be a definite reduction scheme which will outperform PV-LR and be the optimal reduction scheme for this problem. The best reduction scheme for this problem is the PV-LR. It should be noted that all the PV schemes outperformed the plague operator, because the diversity in population and the number of individuals in the PV scheme is higher than the plague operator scheme. It is interesting to note that the computational effort of PV-RAN is quite close to the computational effort of the best reduction scheme, i.e. PV-LR. It is believed that the reduction and increment schemes have very good side-effects on the GP performance, because reduction tends to eliminate poor performing individuals and population increment tends to inject new high performing individuals, where the reduction and incrementing of individuals is centred around a sub-optimal population. The best performing scheme in terms of computational effort, for this problem, is the PV-LI. This highlights the fact that a large population size after the individuals have evolved, containing high fitness values, has a much better impact than a large population size initially where almost all the individuals are performing poorly.

The AES compared with SGP, R_λ, can also be observed in Figure 6.9. The speed of convergence was considered secondary in relation to the computational effort required. However, if any two algorithms perform similarly with respect to computational effort, the AES can be used to evaluate their superiority, i.e. the faster the convergence towards a solution (lower number of generations required), the more superior the algorithm. It can be observed that the AES follows a similar trend with that of the computational effort as far as all the PV-Reduction schemes are concerned. The AES of the plague operator is worse than for SGP, because it suffers from the continuous shortage of population as compared with other algorithms. The PV scheme again demonstrated its superiority over the plague operator in this regard. PV-LR also shows favourable results over the PV-RAN scheme as it results in faster convergence towards a solution. PV-LI will naturally converge later than the PV-Reduction schemes, because it initially contains a small population and it takes time before the populations are rich enough to contribute to the convergence of the GP. All the algorithms except the plague operator scheme outperformed SGP in terms of SR. PV-LR and PV-ER demonstrated the best SR amongst the algorithms discussed in this study for this problem. Overall, the PV-LI can be considered to be the superior algorithm amongst the algorithms discussed in this study for this problem. PV-LR was the best algorithm amongst the reduction algorithms. Figure 6.10 shows a typical result for the degree-4 polynomial regression problem.

```
(+ (* (% (* (+ X (* X X ) ) (* X X ) ) (+ X X ) )
(+ X (+ X X ) ) ) X )
```

Figure 6.10 A typical result for the degree-4 polynomial regression problem

Figure 6.11, Figure 6.12 and Figure 6.13 show the average worst fitness, average mean fitness and the average best fitness for the degree-4 polynomial regression problem for SGP, PV-LR, PV-LI and Plague versus generation. These plots were additionally used to investigate any possible trends between the various algorithms and their impact on fitness.

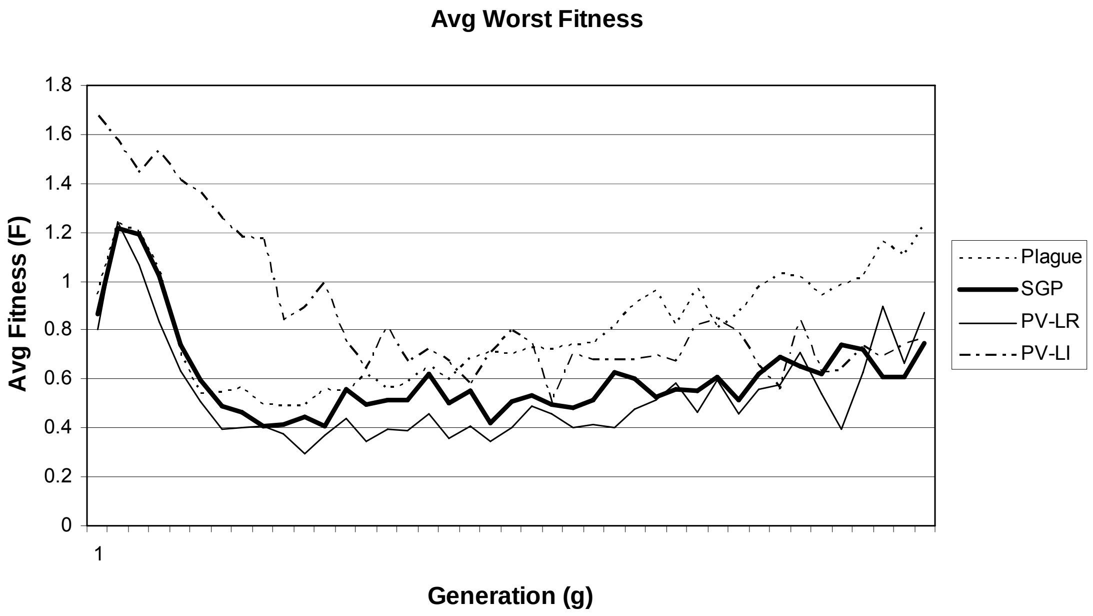


Figure 6.11 Average Worst Fitness

Figure 6.11 suggests that there is no possible trend for the worst performing individuals and that there may be a random variation. Figure 6.12 shows very similar variations for the mean population. Figure 6.13 illustrates the superiority of PV-LR over the other algorithms considering its low computational effort and fast convergence (Figure 6.9).

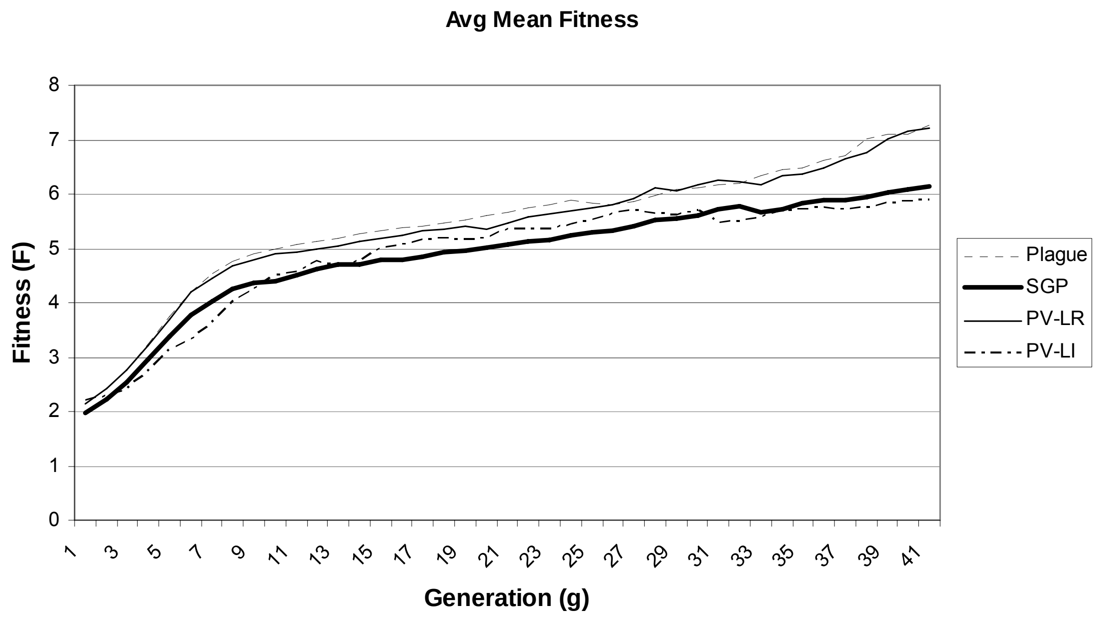


Figure 6.12 Average Mean Fitness

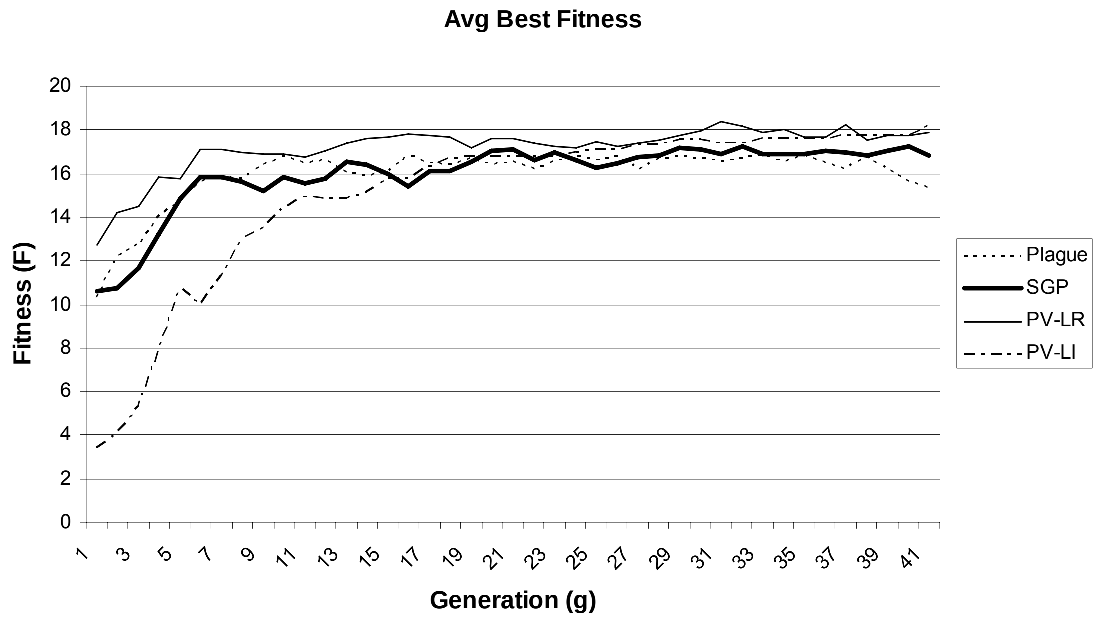


Figure 6.13 Average Best Fitness

PV-LI demonstrates similar performance to PV-LR after a significant number of generations has passed and the population has matured. Figure 6.13 is more important than

Figure 6.11 and Figure 6.12, as it is the best performing individual that determines the success predicate, i.e. final solution. Figure 6.14 shows the graph of the standardised fitness versus computational effort for this problem.

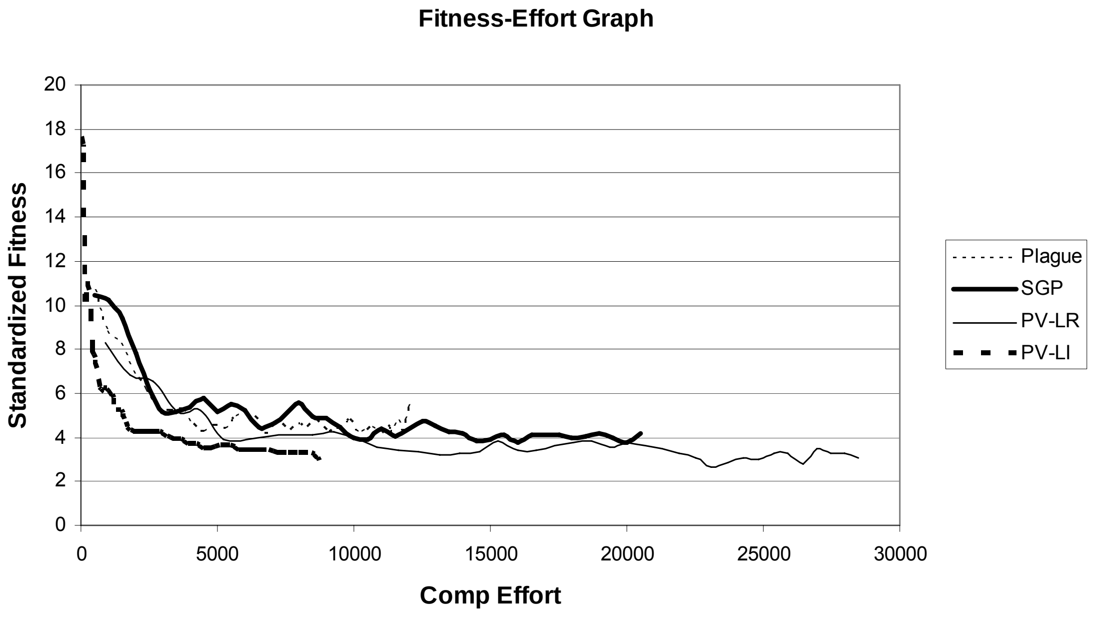


Figure 6.14 Standardised Fitness versus Computational Effort

In summary, Population Variation outperforms SGP and PV-LI was the most effective PV scheme for this problem and PV-LR was the best algorithm amongst the reduction algorithms.

6.5.1.2 Simple Symbolic Regression (degree-3)

The simple symbolic regression (degree-3) problem is considered to be much simpler in comparison with the previous regression (degree-4) problem due to the lower degree of the polynomial and the fact that the GP is not required to create any constants for the coefficients. Figure 6.15 shows the computational effort ratio, the AES ratio and the SR with respect to SGP and Figure 6.16 shows a typical result for degree-3 polynomial regression problem. It was observed that all the population variation reduction schemes, including the plague operator scheme, outperformed the SGP. In addition, the PV schemes proposed in this work achieved better results than the plague operator for this problem. The amount of reduction in population amongst the variants was in the following order: PV-QR, PV-SR, PV-LR and PV-ER. The AES for this problem was approximately ten times better than the AES of the degree-4 polynomial problem due to its simpler structure. Consequently, all the reduction schemes, operating close to their optimal population size, quickly arrived at the solution with a very high SR. Therefore, all the algorithms performed similarly in terms of SR. Additionally, the reduction in population size removed the poor performing individuals and thereby reduced the computational effort.

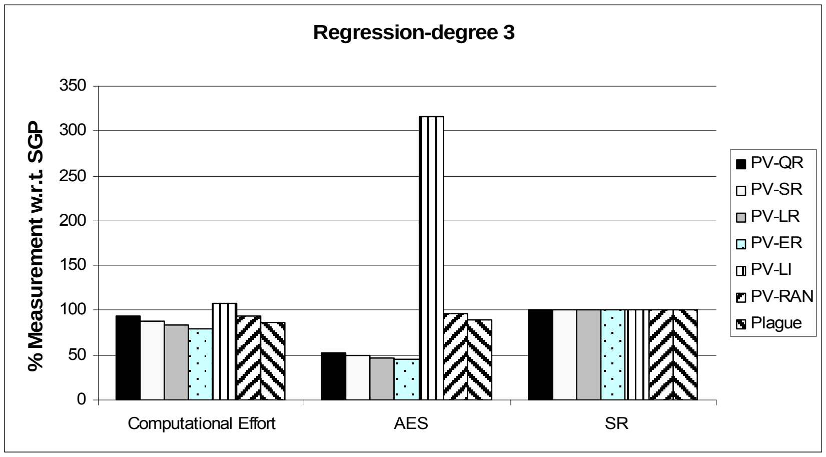


Figure 6.15 Performance Measures with respect to SGP for the degree-3 regression problem

(* (+ (- (% X X) (* X X)) X) X)

Figure 6.16 A typical result for the degree-3 polynomial regression problem

It can be argued that PV-LI scheme is not suitable or cannot compete against the reduction schemes when the population converges rapidly and has a very small λ and high SR. To illustrate, λ ranged from 38 to 84 in the degree-4 polynomial problem, but was between 4 and 8 for this degree-3 problem. By the time PV-LI reached a substantial population size with mature and rich individuals suitable for convergence, it was considerably behind in terms of computational effort when compared with the other schemes, where they had already converged way before this point. The AES followed a similar trend with that of the computational effort. The AES of the PV-LI was the worst amongst the other algorithms as explained in the previous subsection. The PV scheme again demonstrated its superiority over the plague operator in this regard. Overall, the PV-ER was considered to be the superior algorithm amongst the algorithms discussed in this study, for this problem.

6.5.2 Digital Logic Problems

6.5.2.1 Even Parity 4 Generator

Figure 6.17 shows the total computational effort, AES and SR for the even parity 4 generator problem in comparison with the SGP. It can be observed that all the reduction population variants perform equally as well as the SGP, as far as the AES and SR are concerned. Therefore, as they are converging similarly towards a solution, one would expect to have a higher computational effort for these because these algorithms started off initially with a higher population than SGP and hence an extra computational burden. This is clearly seen in Figure 6.17.

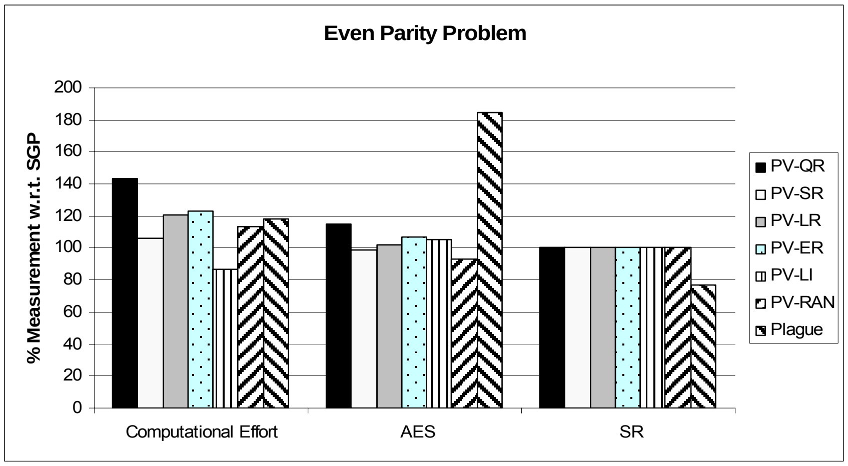


Figure 6.17 Performance Measures with respect to SGP for the even parity 4 generator problem

It can be concluded that for this specific problem the PV-R schemes have not improved the AES and SR and hence are not considered to be effective. It is argued that the fitness landscape is fairly plateaued and this is the reason for such poor performance. As there is a fairly plateaued fitness landscape, AES should be similar for the PV-R schemes.

If the fitness landscape was completely plateaued, then the GP search algorithm would be completely eliminated, because the GP algorithm would approach a blind random search. Consequently, it can be argued that population reduction is not going to be beneficial for such problems and more effort would be introduced, which is contrary to the statement in Fernandez *et al.* [79]. It can be seen that the PV-RAN can lead to faster convergence at an extra cost of computational effort. It is believed that this is due to the beneficial side effects that are introduced as a result of population variation (Please refer to previous section). Plague performs particularly poorly, not only in relation to high computational effort but also at a cost of a slow convergence and worse SR. It is believed that the extra effort is due to the lack of individuals for the GP algorithm to effectively perform under its extreme sub-optimal value, wasting resources and hence resulting in a very slow convergence with a poor SR. It should be noted that the PV-LI variant outperforms the SGP and again demonstrates its superiority over all the other algorithms. Figure 6.18 shows a typical result obtained for the even parity 4 generator problem together with its schematic diagram in Figure 6.19.

(NAND (NOR (NOR (NOR (NAND Y (NAND Y (NOR Z Y))) (NOR (NOR (NAND W X) (NAND W X)) (NOR W X))) (NAND (NAND (NOR (NOR X W) Y) (NAND W X)) Z)) (NOR W (NAND (NAND Z Z) (NOR (NAND (NAND (NAND Z W) X) (NAND W X)) (NAND Y Y)))))) (NAND (NAND (NAND W (NOR X W)) (NOR (NOR (NOR X (NAND Y Y)) Z) (NOR (NAND (NOR (NOR W Z) Z) Z) (NAND W X))))) (NAND (NAND W (NOR X Y)) (NAND (NAND (NAND (NOR (NOR X W) Y) (NAND W X)) (NOR X W)) (NAND (NAND (NAND W X) (NOR (NOR (NOR W Z) (NAND W X)) (NOR Z (NAND Y Y))))) (NAND (NAND (NAND W X) Y) Y)))))))

Figure 6.18 A typical result for the even parity 4 generator problem

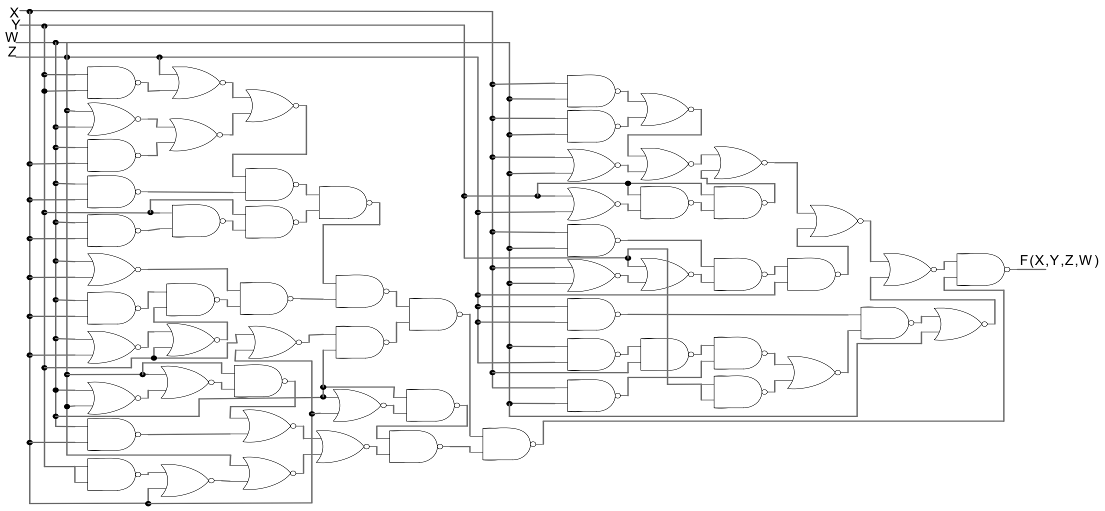


Figure 6.19 The schematic of the solution to the parity 4 generator problem

Figure 6.20, Figure 6.21 and Figure 6.22 show the average worst fitness, average mean fitness and the average best fitness of SGP, PV-LI and Plague versus generation for the even parity 4 generator problem. These plots were also used to investigate any possible trends between the various algorithms and their impact on fitness.

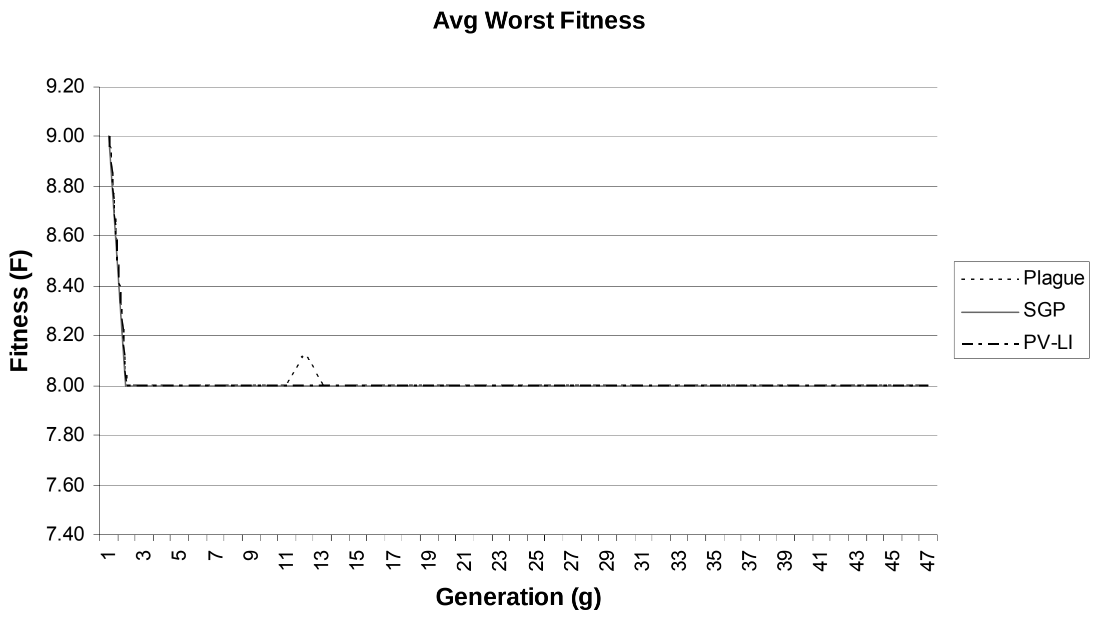


Figure 6.20 Average Worst Fitness

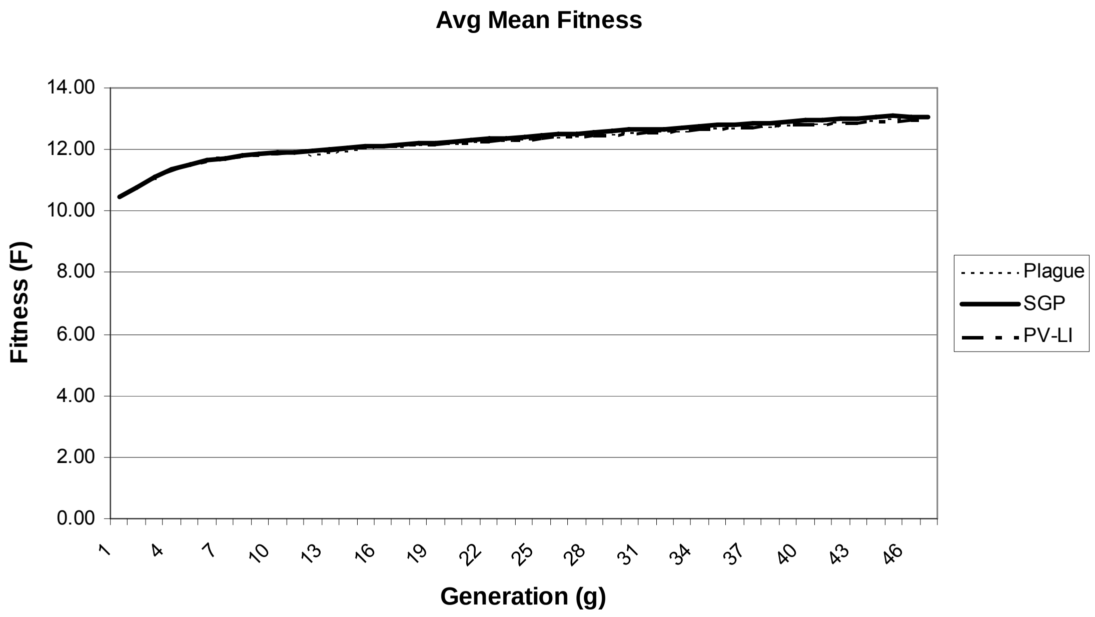


Figure 6.21 Average Mean Fitness

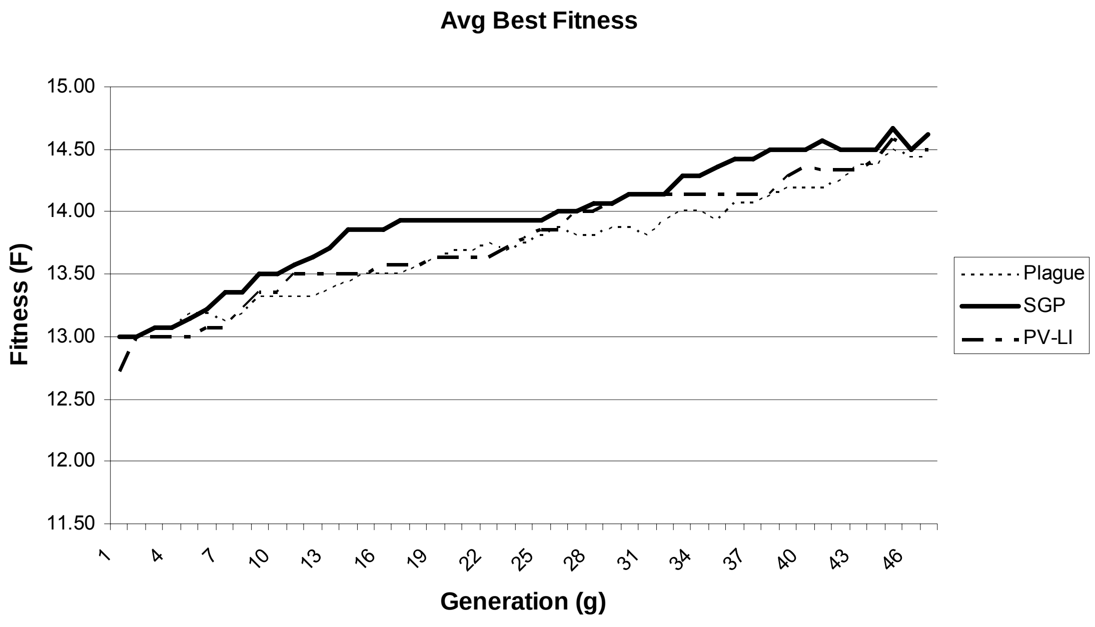
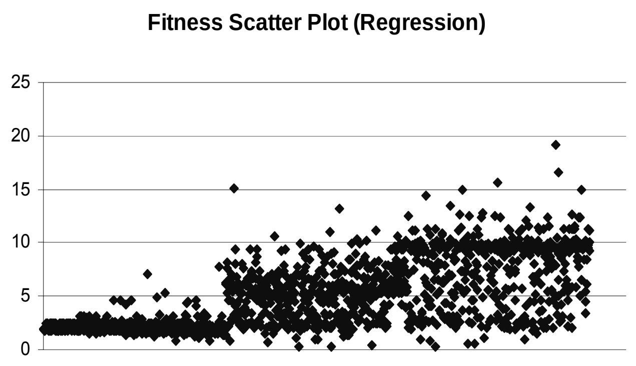


Figure 6.22 Average Best Fitness

From the above Figures, it can be observed that the fitness landscape is fairly flat when compared with the regression problems and that the three algorithms perform almost the same as each other. To illustrate, the percentage change in average best performing fitness and average mean performing fitness is 12.5% and 24.9% respectively for the SGP in this problem, whereas it is 54.7% and 231.9% for the degree-4 regression problem. The percentage change for average best fitness to average worst fitness is 60.4% for SGP for this problem but 1823.4% for the degree-4 regression problem. Figure 6.23 show the scatter plot of fitness for all the individuals from a sample population of initial, midpoint and final generation in a randomly selected run for SGP for the regression problem and the parity problem respectively. In Figure 6.23, attention should be paid to the vertical dispersion of data. Table 6.1 summarises the measures of spread or dispersion of this data.



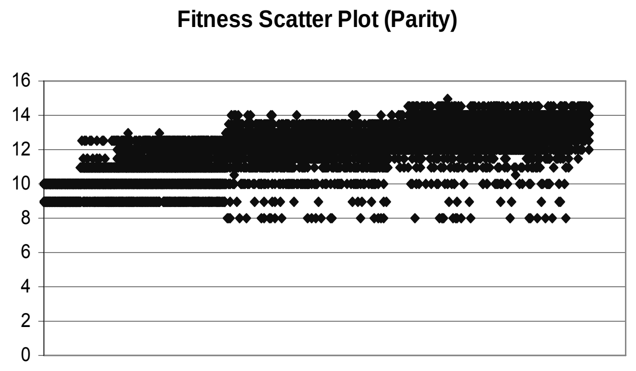


Figure 6.23 Fitness scatter plot – regression and parity problem

Dispersion	REGRESSION PROBLEM (DEG 4)	Parity Problem
Range	18.95	7
Variance	10.95	1.94
Standard Deviation	3.19	1.39

Table 6.1 Measures of Dispersion for a sample data

The analysis above was conducted to compare the fitness landscape between the two problems and to confirm our hypothesis. Figure 6.24 shows the graph of the standardised fitness versus computational effort for this problem.

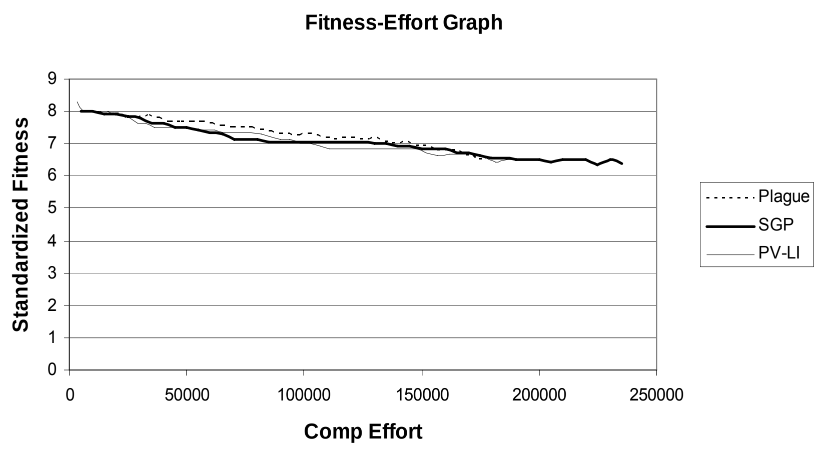


Figure 6.24 Standardised Fitness versus Computational Effort

PV-LI performs better than the Plague and also better than the SGP as the population starts to converge. In summary, Population variation does not perform well when fitness landscape is flat, as search approaches a random blind search and PV-LI only tends to show slight positive effects when compared with SGP under these circumstances.

6.5.2.2 Boolean Symmetry Function

Figure 6.25 shows the total computational effort, AES and SR for the Boolean 6-symmetry problem in comparison with SGP. In this problem, all the population variation reduction schemes, including the plague operator, outperformed the SGP.

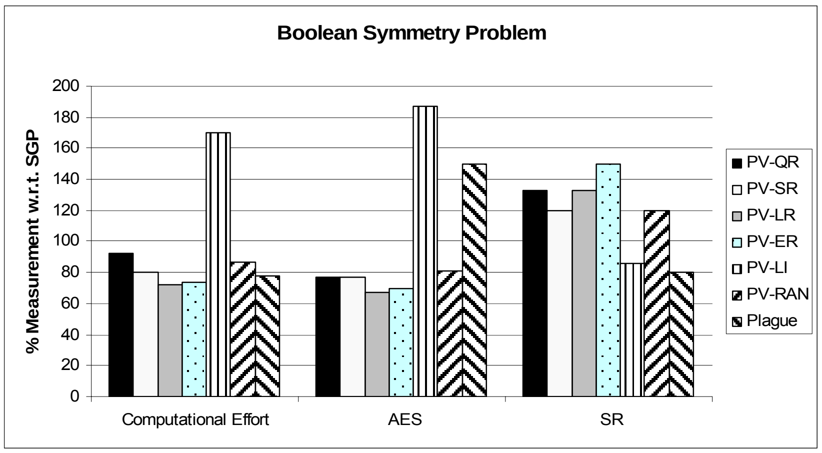


Figure 6.25 Performance Measures with respect to SGP for the Boolean 6-symmetry problem

In addition, all of the PV reduction schemes proposed in this study achieved better results than the plague operator. The amount of reduction in population amongst the variants was in the following order: PV-QR, PV-SR and PV-LR. It can be observed that the AES follows a similar trend with the computational effort. PV-ER and PV-LR had the highest SR for this problem. The PV reduction schemes again demonstrated their superiority over the plague operator in this regard. Overall, the PV-LR was the superior algorithm amongst the algorithms discussed in this study for this problem. Figure 6.26 shows a typical result obtained together with its schematic diagram shown in Figure 6.27.

```
(AND (AND (XNOR (XNOR Y Z ) W ) (AND
(XNOR X P ) K ) ) (AND (XNOR (AND (AND K
(XNOR Y Z ) ) (OR (AND Y (XNOR Y Z ) )
Z  ) ) W ) Y ) )
```

Figure 6.26 A typical result for the Boolean 6-symmetry problem

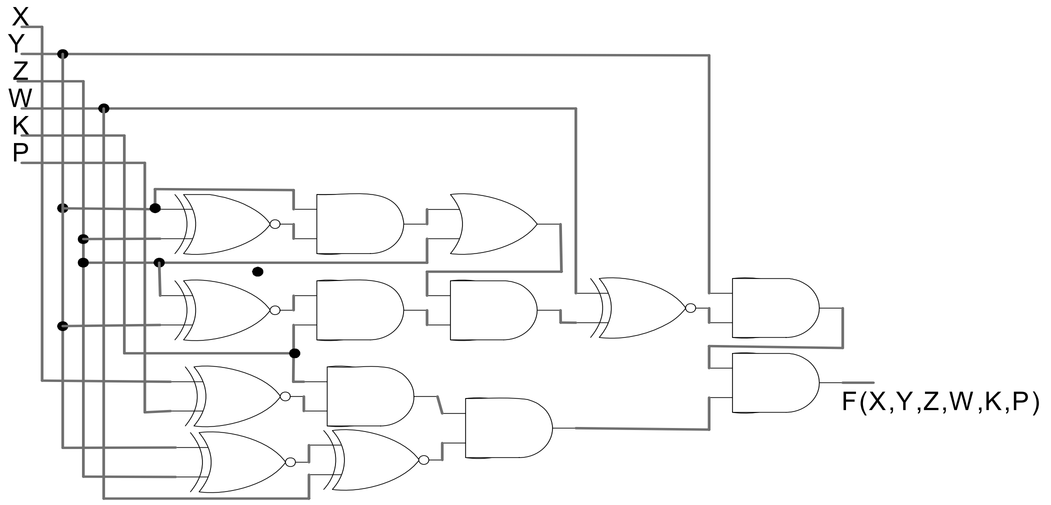


Figure 6.27 The schematic of the solution to the Boolean 6-symmetry problem

6.5.3 Sequence Induction Problems

6.5.3.1 Recursive Sequence Induction

Figure 6.28 shows the total computational effort, AES and SR for the recursive induction problem in comparison with SGP.

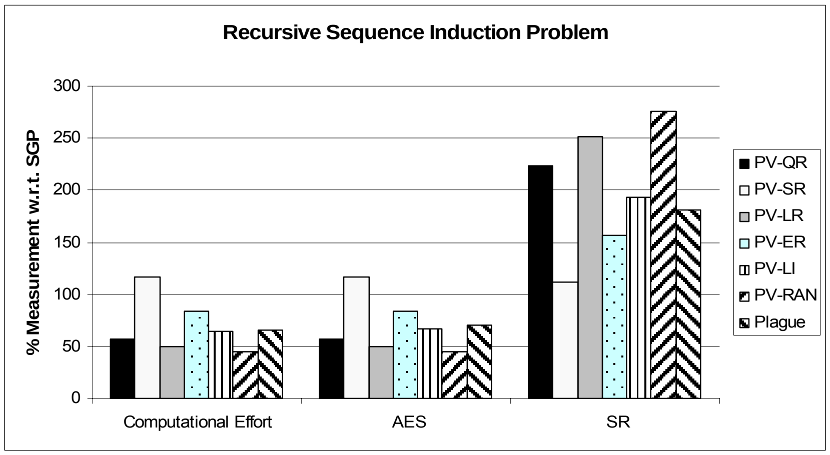


Figure 6.28 Performance Measures with respect to SGP for the Recursive Sequence Induction Problem

It can be observed that all the population variation reduction schemes excluding the PV-SR and the plague operator outperform the SGP. In addition, it can be seen that the best schemes are the PV-RAN and PV-LR algorithms. It can also be observed that the AES follows a similar trend to that of the computational effort. PV-RAN had the best SR. It is believed that the reduction and increment schemes have very good side-effects on the GP performance, as reduction tends to eliminate poor performing individuals and population increment tends to inject new high performing individuals, where the reduction and incrementing of individuals is centred around a sub-optimal population. PV-RAN was effective in solving this problem. Figure 6.29 shows a typical result obtained for this problem.

```
(- J (SRF (* (SRF (- J 1 ) (SRF 1 1 ) ) (SRF (SRF (- J 1 ) J
) (- J 1 ) ) ) (* (- (- J (SRF (- (- J 1 ) 1 ) (- J 1 ) ) ) (SRF (- J
1 ) (- J 1 ) ) ) (SRF 1 (* (- J 1 ) (- J 1 ) ) ) ) ) ) )
```

Figure 6.29 A typical result for the recursive induction problem

6.5.3.2 Simple Sequence Induction

Figure 6.30 shows the total computational effort, AES and SR for the simple sequence induction problem in comparison with SGP.

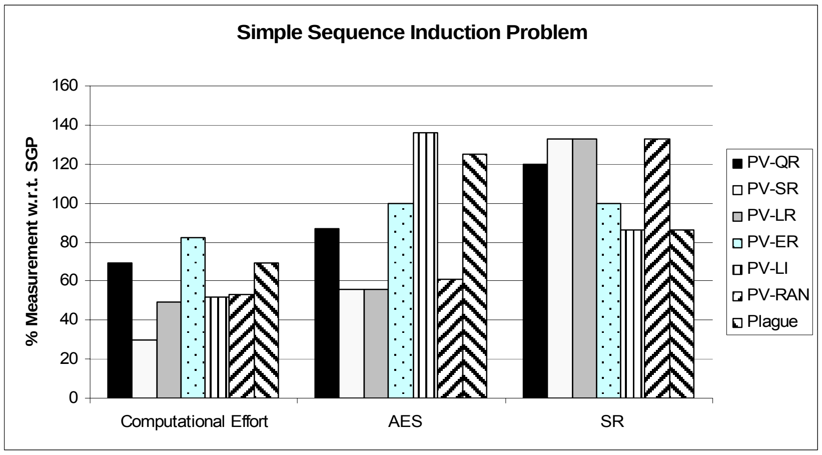


Figure 6.30 Performance Measures with respect to SGP for the Simple Sequence Induction Problem

All the population variation reduction schemes together with the plague operator outperformed the SGP in this problem. It was also observed that all of the PV reduction schemes proposed in this study achieved better results than the plague operator. Moreover, the best scheme was the PV-SR algorithm for this problem. Figure 6.31 shows a typical result obtained for this problem.

```
(+ (* (* J 2 ) 1 ) (+ (+ (+ (* (- J 1 ) (- (* J 2 ) (+ 2 J ) ) ) (+
(* 2 J ) J ) ) 0 ) 1 ) )
```

Figure 6.31 A typical result for the simple sequence induction problem

6.5.4 Artificial Ant Problem

6.5.4.1 Santa Fe Trail

Figure 6.32 shows the total computational effort, AES and SR for the Santa Fe Trail problem in comparison with SGP.

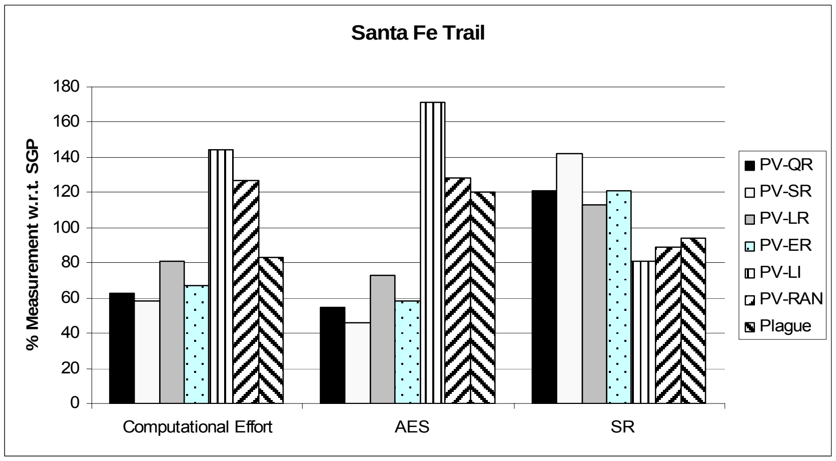


Figure 6.32 Performance Measures with respect to SGP for the Santa Fe Trail

It can be observed that all the population variation reduction schemes together with the plague operator outperform the SGP. Therefore, our results agree with Fernandez *et al.* [79] for the amount of computational effort saved when using plague over SGP. In addition, it can be seen that all of the PV reduction schemes proposed in this study achieved better results than the plague operator for this problem. Moreover, it can be seen that the best scheme is the PV-SR algorithm for the Santa Fe Trail problem. Figure 6.33 shows a typical result obtained for the Santa Fe Trail problem.

```
(( PROGN ) (IF-FOOD-AHEAD (IF-FOOD-AHEAD (
MOVE ) ( MOVE ) ) (( PROGN ) ( RIGHT ) ( MOVE ) )
) (( PROGN ) (( PROGN ) ( LEFT ) (( PROGN ) ( RIGHT
) (( PROGN ) (IF-FOOD-AHEAD ( RIGHT ) ( RIGHT ) )
(IF-FOOD-AHEAD ( MOVE ) ( LEFT ) ) ) ) ) ( LEFT ) )
)
```

Figure 6.33 A typical result for the Santa Fe Trail

6.5.4.2 Santa Fe Trail

All the population variation reduction schemes and the plague operator outperform the SGP for this problem, as shown in Figure 6.34. The best scheme is the PV-RAN algorithm.

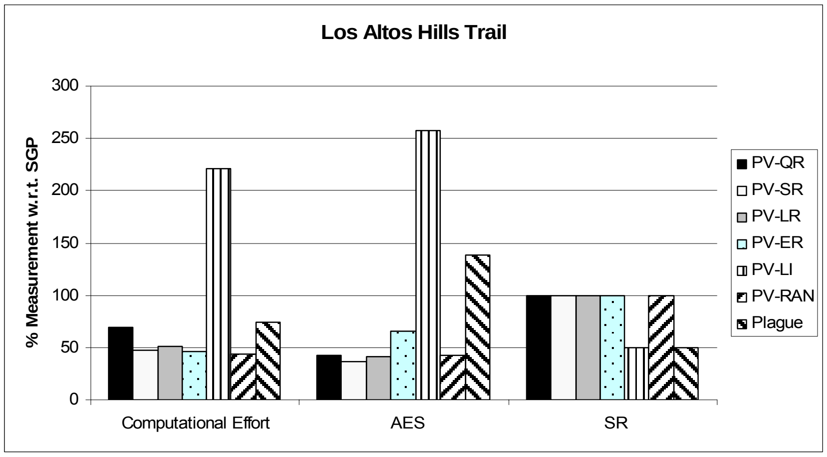


Figure 6.34 Performance Measures with respect to SGP for the Los Altos Hills Trail problem

It can be observed that all the population variation reduction schemes and the plague operator outperform the SGP for this problem and that the AES follows a similar trend with the computational effort. PV-LI and the plague operator showed poor SR and AES performance. The PV-SR, PV-LR and PV-ER reduction schemes performed almost equally and can be considered as the best reduction schemes for this problem.

Figure 6.35 shows a typical result obtained for the Los Altos Hills Trail problem.

```
(PROGN3 ( MOVE ) (PROGN (PROGN ( RIGHT ) (
MOVE ) ) ( LEFT ) ) (PROGN4 ( RIGHT ) (IF-FOOD-
AHEAD (PROGN ( RIGHT ) (PROGN4 (PROGN4 (
LEFT ) ( MOVE ) ( LEFT ) ( RIGHT ) ) (IF-FOOD-
AHEAD ( RIGHT ) (PROGN (PROGN4 (PROGN3 (
LEFT ) ( RIGHT ) ( LEFT ) ) (PROGN (PROGN (
RIGHT ) ( MOVE ) ) ( LEFT ) ) ( MOVE ) (PROGN3 (
LEFT ) ( RIGHT ) ( RIGHT ) ) ) ( LEFT ) ) ) (IF-FOOD-
AHEAD ( RIGHT ) ( LEFT ) ) ( LEFT ) ) ) (IF-FOOD-
AHEAD (PROGN3 (PROGN4 (PROGN ( RIGHT ) (
RIGHT ) ) (PROGN ( LEFT ) (IF-FOOD-AHEAD
(PROGN ( RIGHT ) ( LEFT ) ) (PROGN ( RIGHT ) (
RIGHT ) ) ) ) ( MOVE ) (IF-FOOD-AHEAD (PROGN3
( RIGHT ) ( RIGHT ) (PROGN (PROGN3 (PROGN3 (
RIGHT ) ( RIGHT ) (PROGN4 ( LEFT ) ( MOVE ) (
MOVE ) ( MOVE ) ) ) (IF-FOOD-AHEAD ( LEFT ) (
LEFT ) ) ( LEFT ) ) (PROGN3 (IF-FOOD-AHEAD (IF-
FOOD-AHEAD (PROGN4 ( LEFT ) ( RIGHT ) ( MOVE
) ( LEFT ) ) (PROGN ( RIGHT ) ( RIGHT ) ) ) (PROGN
( RIGHT ) ( RIGHT ) ) ) (PROGN ( RIGHT ) ( RIGHT )
) ( LEFT ) ) ) ) (IF-FOOD-AHEAD (IF-FOOD-AHEAD
(PROGN ( RIGHT ) ( LEFT ) ) (PROGN ( RIGHT ) (
RIGHT ) ) ) (PROGN ( RIGHT ) (PROGN ( RIGHT ) (
LEFT ) ) ) ) ) ) (PROGN4 ( RIGHT ) (IF-FOOD-
AHEAD (PROGN4 ( LEFT ) ( RIGHT ) ( MOVE ) (
LEFT ) ) (IF-FOOD-AHEAD (IF-FOOD-AHEAD (IF-
FOOD-AHEAD (PROGN4 ( LEFT ) ( RIGHT ) ( MOVE
) ( LEFT ) ) (PROGN3 ( LEFT ) ( RIGHT ) ( RIGHT ) ) )
(PROGN (PROGN4 ( MOVE ) ( LEFT ) ( LEFT ) (
RIGHT ) ) ( LEFT ) ) ) (PROGN ( RIGHT ) ( RIGHT ) )
) ) ( MOVE ) (IF-FOOD-AHEAD ( MOVE ) ( RIGHT ) )
) ( LEFT ) ) (PROGN ( RIGHT ) ( RIGHT ) ) ) ) ) (
MOVE ) (IF-FOOD-AHEAD ( MOVE ) ( RIGHT ) ) ) ) )
```

Figure 6.35 A typical result for the Los Altos Trail

6.6 Conclusion and Finishing Remarks

In this chapter the concept of population variation was presented and it was investigated whether the nature of the "population variation", i.e. the way the population is altered during the run, has any significant impact on GP performance in terms of computational effort. The main aim was to save computational effort by systematically altering the population size (by either increasing or decreasing the population size or performing both) by various strategies during the run. The faster convergence to a solution and SR was considered a bonus. We experimented with various schemes for population variation on four different problems on two separate landscapes for each problem to provide a good range of problem diversity.

Experimental evidence has been produced to show that population variation can significantly improve performance by reducing the computational effort to reach a given fitness level, provided that the fitness landscape is not plateaued. Population reduction only works when the fitness landscape is not plateaued. It was also shown that the way the population is reduced has an impact on performance depending on the problem domain, which is ultimately driven by the fitness landscape. Consequently, every specific problem has its own most favourable reduction profile. It was illustrated that the best performing profiles studied herein were the linear increment scheme for the degree-4 symbolic regression problem, exponential reduction scheme for the degree-3 regression problem, linear increment scheme for the parity problem, linear reduction scheme for the Boolean symmetry problem, random reduction and step reduction for the recursive sequence induction and simple induction respectively, step reduction for the Santa Fe trails and random reduction schemes for the Los Altos Hills trail.

Evidence was also produced to show that the proposed PV scheme achieved better results than the plague operator scheme and outperformed the SGP. In the next chapter the dynamic adaptation of population size and its impact on the GP algorithm using a heuristic feedback mechanism is investigated and it is shown to result in further improvements.

CHAPTER 7

DYNAMIC POPULATION VARIATION

"Everyone is a genius at least once a year. The real geniuses simply have their bright ideas closer together.", Frank Lloyd Wright.

7.1 Introductory Notes

In the previous chapter, the concept of population variation (PV) scheme was introduced, where the population could be increased and/or decreased with a variable profile, where the increment or reduction of population size could take on any flexible profile such as linear, exponential, hyperbolic, sinusoidal or even random. The population size was varied during the run and it was demonstrated that PV significantly improved performance and showed that the optimum profile was dependant on the problem domain, which was ultimately driven by the fitness landscape. This investigation [164] involved what can be referred to as static population variation in genetic programming. Static population variation employs a deterministic adaptation approach using simple time-varying schedules, where the population size is varied according to a deterministic function $P(g)$. The function $P(g)$ is the Population-Generation profile, which determines the total number of individuals (the population size) at any generation g. One of the shortcomings of the static population variation scheme is that the population size is varied by a blind deterministic function. It is more desirable to vary the population size in an informed way during the run. Using a heuristic feedback mechanism, the population size can be dynamically varied by taking into account the actual progress of GP in solving the problem. In this chapter another technique is investigated, where the size of the population is varied dynamically during the execution of the GP system. The population size is varied "on the run" according to some particular events occurring during the evolution. Table 7.1 clearly defines and summarises the difference between population variation and dynamic population variation.

Algorithm	Acronym	Description
Population Variation	PV	It employs a deterministic adaptation using simple time-varying schedules. The population is changed using a deterministic function. The population profile is known prior to the run, hence termed as "static".
Dynamic Population Variation	DPV	It employs a feedback mechanism. By using the actual progress made, the population profile is "dynamically" changed during the run. The population profile is unknown prior to any given run.

Table 7.1 Population Variation vs. Dynamic Population Variation

In this section, various new ways are proposed to dynamically vary the population size during the run of the GP system. The proposed approach referred to as Dynamic Population Variation (DPV) [165], extends the work of Fernandez *et al.* [81] and proposes new modifications to it.

7.2 Introduction to Dynamic Population Variation

As described in the previous section, it is desirable to dynamically vary the size of the population during the execution of the GP system. The population size is dynamically increased if the change in fitness improvement is less than some functional value, called the pivot ρ as defined by (7.1) to (7.3), and reduced if this change is more than the pivot. The motivation is to add new individuals when the GP system is reaching a stagnation phase and remove individuals when the GP process is progressing well. Two specific pivot functions, DIV and SUP were introduced [81] with the dynamic modification of the population size described as a pseudo-algorithm. Herein, the modifications are summarised by (*7*.1) to (*7*.3), showing the respective calculations of the pivot functions.

$$\rho_{DIV}^{<j,j+T-1>} = \frac{1}{T} \times \sum_{i=j}^{j+T-1} \begin{cases} \frac{\Delta_{i-1}}{\Delta_i}, & \Delta_i \neq 0 \\ 1, & \Delta_i = 0 \end{cases} \qquad (7.1)$$

$$\rho_{SUP}^{<j,j+T-1>} = \frac{1}{T} \times \sum_{i=j}^{j+T-1} (\Delta_{i-1} - \Delta_i) \qquad (7.2)$$

$$\Delta_i = f_{i-1} - f_i \qquad (7.3)$$

Where,

f_i is the fitness of the best individual in the population at generation *i*.

T is the period defined as the number of generations over which the pivot is calculated. In other words, the period is the number of generations between two successive updates of the pivot.

It should be noted that the period *T* can be set to 1, forcing the pivot to be renewed at each generation. In this study, the period has been set to 1 to renew the pivot at each generation. Population size is varied at each generation according to (*7*.4), which defines the change in population size at generation *g*. The generation number "*g*" ranges from 0 (the initial generation containing the initial random population) to *G*, which is the maximum number of generations to be run. *P*(*g*) is the total number of individuals (the population size) at any generation *g*.

$$\Delta P(g) = \begin{cases} +0.2\% \times P(g) \times \Delta_g, & \Delta_g < \rho \\ -1\% \times P(g) \times \Delta_g, & \Delta_g \geq \rho \end{cases} \qquad (7.4)$$

Given the set of trees $\boldsymbol{T} = \{T_1, T_2, T_3, \ldots, T_M\}$, which form the M populations at generation g, sorted according to fitness $f(T_1) \leq f(T_2) \leq f(T_3) \leq \ldots \leq f(T_M)$, where f is the fitness function, the k worst trees from a fitness point of view are removed at each generation for decrementing the population size. For incrementing the population size, the m best performing trees from a fitness point of view are selected and mutated at each generation. These mutated individuals are then added to the next generation.

In this chapter, three possible modifications to this technique of dynamic population variation are explored and investigated, as outlined in each of the subsections 7.2.1 through to 7.2.3. The first modification includes the introduction of four new and different characteristic measures for determining and assessing the stagnation phase. Secondly, a new gradient based pivot function (GRAD) is proposed along with an attempt to determine the role of different pivot functions. The GRAD pivot function is compared with the DIV and SUP pivot functions. Finally, the population change equation (7.4) is modified to remove the special constants within it and make it more general. It is strongly believed that such constants should be avoided if possible, as they can be problem dependant and restrict generalisation. Table 7.2 summarises the proposals that are investigated in this chapter.

Proposals	Description
Formalisation of population variation	The population variation is further formalised via the Dynamic Population Variation (DPV) with its new proposals outlined below and replacement of pseudo-algorithm with appropriate equations.
Stagnation Assessment	Various new models are proposed via the characteristic measure Φ to assess stagnation and alter population size based on this measure.
Pivot Function	The GRAD Pivot Function is introduced and investigated.
Generic Population Variation	A new method to alter population size is introduced eliminating the need for any special constants.

Table 7.2 Summary of Dynamic Population Variation (DPV)

7.2.1 Stagnation Phase Assessment

The stagnation phase within GP as defined in equation (7.3) is measured in terms of the best performing individual in the population at any generation. It is suggested that some different ways of measuring the stagnation phase should be investigated and it is required to be determined whether a more appropriate measure can be defined. This previous definition simply relies on the best performing individual. Consider the box and whisker plot depicted in Figure 7.1, which shows the spread of fitness values for two different generations, generation *i* and generation *j* at some time later in the evolution. The best performing individuals in both generations have the same fitness. However, all the other individuals within generation *j* have improved and are progressing. If the previous definition is used, the GP is considered to be stagnating, which may be considered a contradiction to the observation in Figure 7.1. All the individuals with the exception of the best performing individual have significantly progressed from one generation to the other as shown in Figure 7.1.

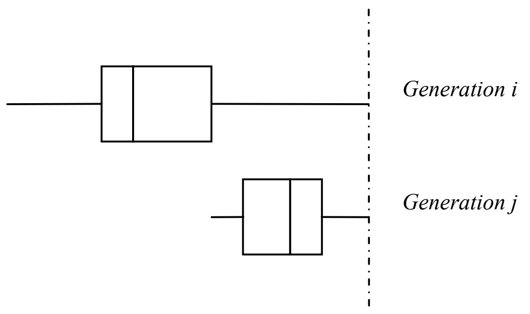


Figure 7.1 Fitness Distribution of two generations - Box and Whisker Plot

A general delta function Δ is defined by (7.5) which computes the difference of a characteristic measure at two different generations. The measure can represent any measurable value, criterion, or characteristic feature. If it represents the fitness of the best performing individual then the generalised delta function is simplified to the original function as defined by (7.3).

$$\Delta_i = \Phi_{i-1} - \Phi_i \qquad (7.5)$$

In this chapter, four possible attributes which Φ can take are investigated. The characteristic measure Φ can represent the mean of the fitness function for all the individuals at previous generation in comparison with the current generation, as defined by (7.6). The performance of all the individuals combined could provide a better indicator of a stagnation phase.

$$\Delta_i = \frac{\sum f_{i-1}}{P(i-1)} - \frac{\sum f_i}{P(i)} = \mu_{i-1} - \mu_i \qquad (7.6)$$

Alternatively, Φ can represent the fitness median *m* between two successive generations as defined by (7.7). The fifty percentile mark may represent a more accurate assessment of the stagnation phase, as extreme values or outliers can significantly affect the mean.

$$\Delta_i = m_{i-1} - m_i \qquad (7.7)$$

The measure Φ can also represent the fitness upper quartile (UPQ) difference between two successive generations as defined by (7.8).

$$\Delta_i = upq_{i-1} - upq_i \qquad (7.8)$$

Finally, the difference of the mean of high performing individuals between two consecutive generations as defined by (7.9) is investigated. The 90% mark is considered as the cut-off for the high performing (HP) individuals. The number of individuals from a fitness perspective within this window (90% cut-off) at generation *g* is represented by $n_{HP}(g)$. $\sum_{HP} f_g$ denotes the sum of fitness values of all the individuals within this range at generation *g*. The performance of all the "top" performing individuals combined could more accurately characterise the stagnation phase. This study should provide a better understanding of the stagnation phase and its definition in GP, at least for the problems studied herein.

$$\Delta_i = \frac{\sum_{HP} f_{i-1}}{n_{HP}(i-1)} - \frac{\sum_{HP} f_i}{n_{HP}(i)} \qquad (7.9)$$

7.2.2 Role of the Pivot Function

The pivot function is used to decide whether new individuals are required to be inserted into the population or existing individuals are needed to be eliminated. It is eventually employed in determining the stagnation phase. Using (*7*.1), (*7*.3) and (*7*.4), it can be shown that the relationship defined by (*7*.10) is present for the DIV pivot function, when the period is set to 1.

$$\Delta_i \geq \sqrt{\Delta_{i-1}} \tag{7.10}$$

Correspondingly, a similar relationship can be portrayed for the SUP pivot function as defined by (*7*.11).

$$\Delta_i \geq \frac{1}{2} \times \Delta_{i-1} \tag{7.11}$$

It was reported in [81] that the DIV pivot function demonstrated a superior performance to the SUP pivot function. This could be the expected behaviour for the best-performing-individual (*7*.3) since fitness improvements for the best performing individual normally reduce rapidly with very small changes. Hence the DIV pivot function reflects a more accurate measurement of the stagnation phase and even counter-affecting it, whereas the DIFF unnecessarily subjects the GP to further constraints by removing individuals. It is believed that the expected performance improvement is a complex function of population size, the individual or criterion in question, the fitness landscape and many other possible variables. To illustrate, the GRAD pivot function is introduced as subjected to the constraints in (*7*.12) and defined by (*7*.13).

$$\frac{d\Delta_i}{di} \geq \frac{d\Delta_{i-1}}{di} \tag{7.12}$$

$$\rho_{GRAD}^{<j,j+T-1>} = \frac{1}{T} \times \sum_{i=j}^{j+T-1} \Delta_{i-1} \tag{7.13}$$

It is expected that the gradient pivot function will perform worse than the DIV and SUP pivot functions when Φ is defined as the best-performing individual, because it subjects GP to more constraints, and thereby opposes the natural intricacy of the genetic programming algorithm in solving solutions. However, it would be interesting to study the performance of the GRAD pivot function for other values of Φ, as the expected performance improvements for the other measurable criteria cannot be as easily formulated as the best performing individual.

It is believed that a pivot function subjected to the “true” GP constraint will yield much better results than the DIV, SUP or GRAD pivot functions. It is also believed that there is an optimum pivot function for each GP problem that describes the true effort required for the specific algorithm that yields the best performance.

7.2.3 Constant Replacement

It is desirable to have a formula that computes the new population size without the special constants defined in (7.4). It is believed that such constants are not pertinent to the GP algorithm and are considered superfluous or inappropriate. Any empirical constant in an algorithm can be problem dependant and will restrict generalisation. Consequently, (7.4) is modified to (7.14) according to a proportionate population increment/decrement scheme without the need for special constants.

$$\Delta P(g) = (-1)^n \frac{\left|\Delta_g - \Delta_{g-1}\right|}{f_{optimal}} \times P(g) \qquad (7.14)$$

Where,

$$n = 1 \quad if\, \Delta_g \geq \rho$$
$$n = 2 \quad if\, \Delta_g < \rho$$

7.2.4 Computational Effort

Similar to the previous chapter the efficiency of the GP algorithm is determined by the amount of computational effort ε expended, as defined by (7.15), i.e., the lower the computational effort, the more efficient the algorithm. The main goal is to minimise the computational effort as indicated by (7.16). The Average number of Evaluations (AES), λ, is considered secondary to the computational effort.

$$\varepsilon_c = \sum_{g=0}^{g=G} P(g) \tag{7.15}$$

$$\varepsilon_{c,DPV} \leq \varepsilon_{c,SGP} \tag{7.16}$$

7.2.5 General Remarks

It is noted that if the fitness is not improving at all, then the SUP pivot functional value is equal to zero, as follows:

$$\begin{aligned} & f_i = f_{i-1} = f_{i-2} = \cdots \\ & \therefore \Delta_i = \Delta_{i-1} = \Delta_{i-2} = \cdots = 0 \\ & \Rightarrow \rho_{SUP} = \frac{1}{T} \times \sum_j \Delta\Delta_j = 0 \end{aligned} \tag{7.17}$$

However, this is not true for the DIV pivot function, even though $\Delta_{i-1} = 0$. In this instance, as Δ_i also evaluates to zero and division by zero is undefined ($\frac{0}{0}$), the pivot will evaluate to one rather than zero as follows:

$$\begin{aligned} & f_i = f_{i-1} = f_{i-2} = \cdots \\ & \therefore \Delta_i = \Delta_{i-1} = \Delta_{i-2} = \cdots = 0 \\ & \Rightarrow \rho_{DIV} = \frac{1}{T} \times \sum_j \frac{\Delta_{j-1}}{\Delta_j} = \frac{1}{T} \times \sum_j 1 = 1 \end{aligned}$$

$$, \text{as } \frac{\Delta_{i-1}}{\Delta_i} = 1, \quad \forall \Delta_i = 0. \tag{7.18}$$

It is noted that a rapidly improving fitness implies that $\Delta_i >> \Delta_{i-1} >> \Delta_{i-2} >> \cdots$. For the DIV pivot function, this means a division by a larger number and hence a smaller pivot. For the SUP pivot function, this means a subtraction from a larger value and hence this could lead to a negative pivot. Figure 7.2 can assist in visualizing this point.

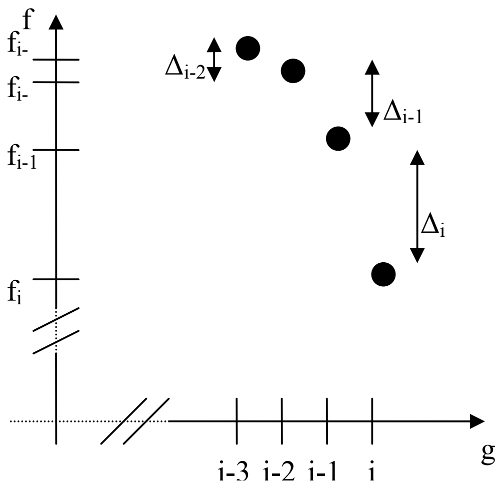


Figure 7.2 Fitness versus generation – Minimisation Problem

It should be noted that these facts contradict the published values of [81], where it was mentioned that if fitness is not improving at all, the pivot will be equal to zero. This is only true for the SUP pivot function as shown above. It was also stated that if fitness is rapidly improving, the value of pivot is high, otherwise it is low. In addition, the following inequality was noted, which is not necessarily correct: $\Delta_{g-1} \geq \Delta_g \geq 0$. Figure 7.2 can be used as a counterexample. Moreover, it was noted that the pivot is always positive [81]. This is only true for the DIV pivot function, but not true for the SUP pivot function, because the inequality $\Delta_{g-1} \geq \Delta_g \geq 0$ does not always hold.

7.3 Experimental Results

The percentage average total computational effort with respect to SGP, or computational effort ratio R_ε is defined by (7.19) and the percentage AES with respect to SGP, R_λ is defined by (7.20).

$$R_\varepsilon = \frac{\varepsilon_{DPV}}{\varepsilon_{SGP}} \tag{7.19}$$

$$R_\lambda = \frac{\lambda_{DPV}}{\lambda_{SGP}} \tag{7.20}$$

7.3.1 Symbolic Regression Problem

Figure 7.3 and Figure 7.4 show the comparative total computational effort and AES results for the degree-4 and degree-3 polynomial regression problems respectively. The SGP algorithm is compared with the GRAD, DIV and SUP pivot functions using the best performing individual as the selected characteristic measure along with standard population increment/decrement according to (7.4). Similar results were obtained for the proportionate population increment/decrement according to (7.14). It can be observed that the AES and the computational effort follow similar trends. As expected, the DIV pivot function outperformed the other pivot functions when the best performing individual was selected as the characteristic measure Φ. This is due to the fact that the DIV function continuously injects new high performing individuals into the population when the progress in terms of fitness of the best performing individual starts to slow down.

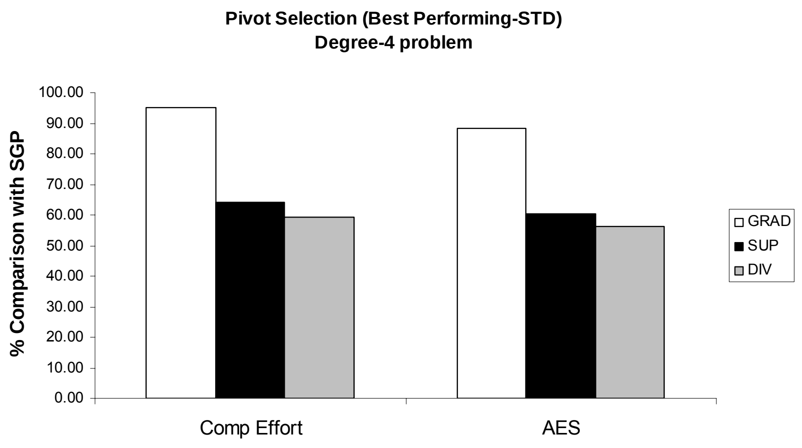


Figure 7.3 Pivot Selection Function – Best Performing Individual with standard increment/decrement (Degree-4)

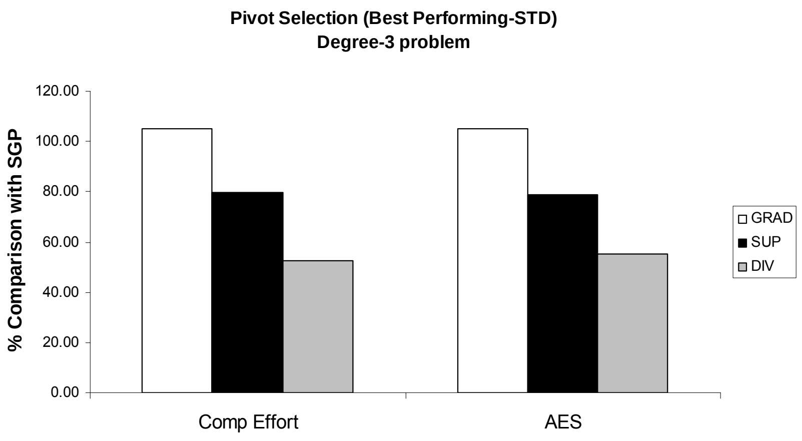


Figure 7.4 Pivot Selection Function – Best Performing Individual with standard increment/decrement (Degree-3)

When the characteristic measure Φ was defined as the top performing individuals (UPQ and HP mean), comparable behaviour was observed as that for the best performing individual. In general, SUP was superior to GRAD in these instances, but DIV was superior to the other two pivot functions when the characteristic measure Φ was the top performing individuals. For this problem, because the fitness landscape is very steep and fitness improvements are large, all pivot functions behaved similarly during the initial phase of GP with respect to population modifications but improvements continually diminished for these characteristics towards the end of the generation. DIV injected new individuals to boost the performance of the GP and thereby attempted to increase the fitness of the population, whereas the GRAD and SUP continually removed individuals in these instances and therefore made it harder for the GP system to converge towards the solution. This is the main reason for DIV's superiority.

For mean and median characteristic measures DIV gave over its superiority to GRAD and SUP. In these cases the focus being on the central values, there was continuous improvement in Φ, but as these were only in small amounts GRAD and SUP tended to remove individuals as a result. However, the DIV pivot function from the initial stages incorrectly considered these to be stagnated and unnecessarily enforced more pressure on the GP to perform by increasing the population. These extra individuals at the initial stages of the iteration become a computational burden and hence resulted in poor performance of DIV in these instances. In general, in most instances the proportionate population increment and reduction scheme according to (*7*.14) performed very similarly to the standard scheme according to (*7*.4). In this case (*7*.14) provided a more generalised approach to population variation. Although there were improvements observed for the proportionate scheme over the standard reduction increment scheme in some instances, in other instances a small performance reduction was observed. The performance reduction was mainly observed during the periods when the changes in population variation according to (*7*.14) resulted in large and abrupt changes, whereas the changes in population as a result of (*7*.4) were always fixed and smooth.

Overall, the best performing pivot function for the degree-4 polynomial problem was the DIV pivot function using the HP mean characteristic measure and proportionate population increment/decrement, as shown in Figure 7.5. However, the best performing pivot function for the degree-3 polynomial problem was the DIV pivot function using the HP mean characteristic measure with fixed or standard population increment/decrement as shown in Figure 7.6. Both of these pivot functions were superior to SGP, the plague operator [79], dynamic population modification technique in [81] and all the static PV schemes reported in [164].

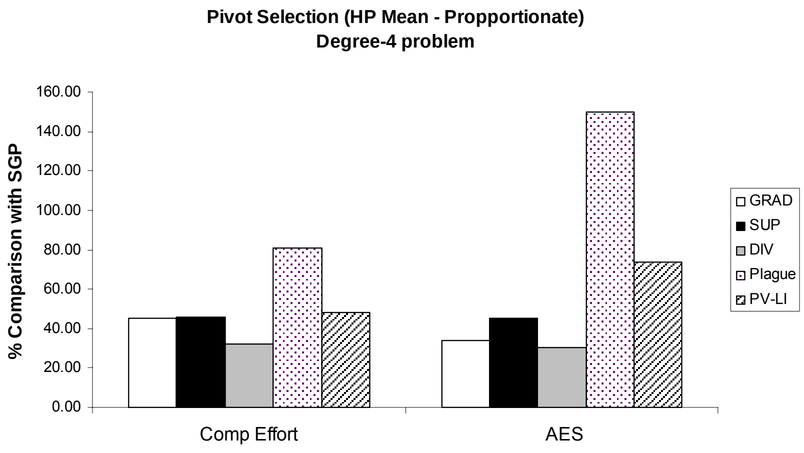


Figure 7.5 Pivot Selection Function – High Performing Mean with Proportionate Increment/decrement)

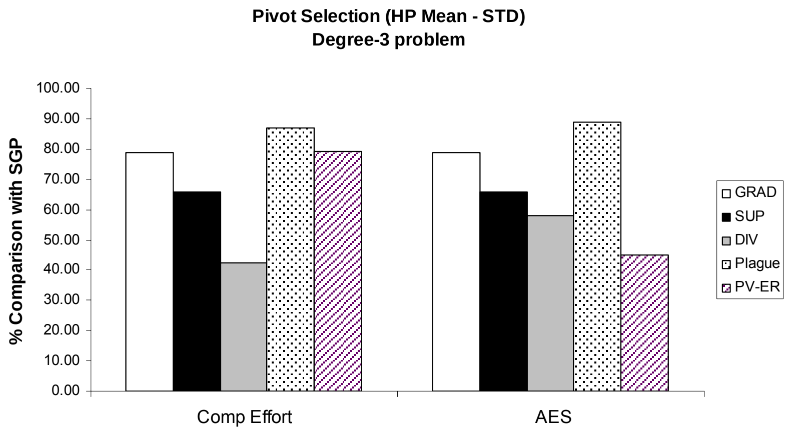


Figure 7.6 Selection Function – High Performing Mean with standard increment/decrement

7.3.2 Digital Logic, Sequence Induction and Artificial Ant Problems

For the even parity 4 generator problem, the GRAD pivot function outperformed the other pivot functions. It is argued that the fitness landscape is fairly plateaued [164] and this is the reason for such performance. If the fitness landscape is completely plateaued then the GP search algorithm would be completely eliminated because the GP algorithm would approach a blind random search. This means that for all the characteristic measures improvements were quite slow during the initial phase and GRAD continually removed individuals as a result, whereas DIV introduced new individuals. However, towards the end of the generation the progress of the characteristic measure Φ significantly deteriorated and individuals were needed to be added in this instance (similar behaviour for all pivot functions here). Although it is beneficial to remove individuals at early stages of the generations both the DIV and SUP pivot functions increased the population during these stages. These extra individuals at the initial stages of the iteration become a computational burden resulting in inferior performance. This observation is in agreement with the findings in [164], where PV-LI (Population Variation Linear Increment) demonstrated its advantage when the population is reduced at the beginning and increased towards the end of the generation. Overall, the best performing pivot function for the even parity 4 generator problem was the GRAD pivot function using the HP mean characteristic measure with standard population increment/decrement as shown in Figure 7.7. GRAD had a smaller computational effort at the cost of higher AES since individuals were removed from the population in the initial stages of GP. On the other hand the AES of the DIV pivot function appeared to be smaller at the cost of higher computational effort due to the addition of extra individuals introduced initially by the DIV pivot function. DIV still outperformed SGP, all the static PV schemes [164], plague [79] and the dynamic population modification technique in [81].

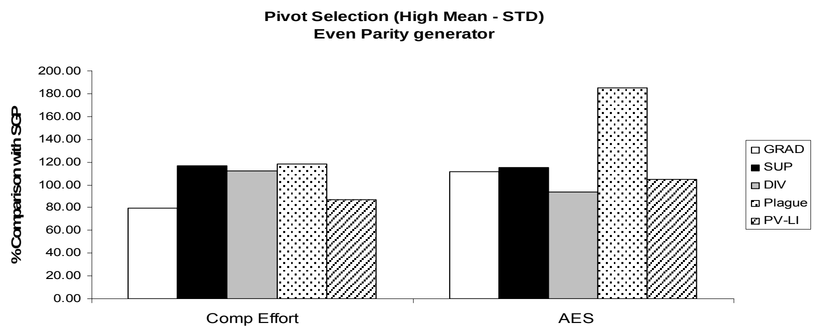


Figure 7.7 Pivot Selection Function – High Performing Mean with standard increment/decrement

For the Boolean 6-Symmetry problem, DIV was the better pivot function for most of the characteristics measures, except for the best performing individual. A similar analysis applies here as for the regression problem and the even parity 4 generator problem. The DIV pivot function using the best performing individual characteristic measure did not perform well because the fitness improvements were extremely slow from the outset. However, for the other characteristics measures there was continuous improvement from the beginning ensuring DIV's dominance in these cases. For the Boolean 6-Symmetry problem, GRAD with upper quartile selection and proportionate population increment/decrement was the best pivot function as shown in Figure 7.8. Similar conclusions as that for the regression problems can be made in relation to the proportionate population increment/decrement. Once more, DPV was superior to the all other previously reported algorithms [79] [81] [164].

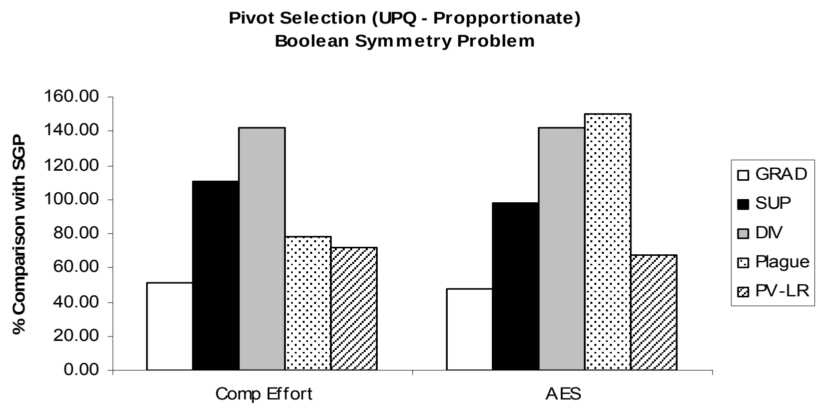
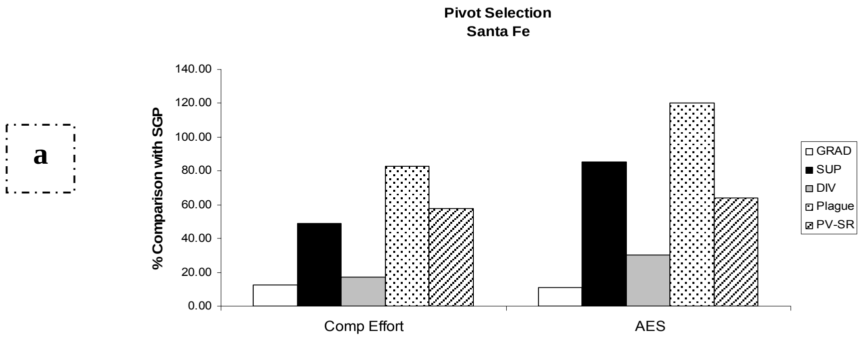


Figure 7.8 Pivot Selection Function – UPQ with Proportionate Increment/decrement

The trends in improvement achieved with the characteristic measures for the Fibonacci sequence problem were similar to the parity generator problem and for similar reasons the SUP pivot function showed its superiority. Also, the best pivot function for the simple sequence induction problem was the DIV pivot function. The SUP pivot function with upper quartile selection and standard population increment/decrement was the best pivot function for the Fibonacci sequence problem, whereas the DIV pivot function using the HP mean as the characteristic measure with standard population increment/decrement was the best pivot function for the simple induction problem. Comparable conclusions can be drawn for the proportionate population increment/decrement as that for the other problems. For the Fibonacci sequence problem both of the pivot functions, SUP with upper quartile selection and DIV with HP mean selection, were superior to SGP, the plague operator [79], the dynamic population modification technique in [81] and all the static PV schemes reported in [164].

The best pivot function for the Santa Fe problem was the GRAD pivot function (Figure 7.9a). Although DIV with top mean selected as characteristic measure with standard population increment and decrement performed exceedingly well, the GRAD pivot function with upper quartile selection and standard population increment/decrement was the best pivot function for this problem. Amongst the proportionate increment/decrement schemes DIV with top mean as characteristic measure was the best. It is pointed out that both the DIV and GRAD with standard increment/decrement outperformed SGP, all the static PV schemes [164], plague [79] and the dynamic population modification technique in [81]. For the Los Altos Hill trail problem, GRAD was the better pivot function for most of the characteristics measures, except for the best performing individual and the mean. The GRAD pivot function with top mean as characteristic measure and standard population increment/decrement was the best pivot function for this problem (Figure 7.9b). Similar trends were observed for the proportionate increment/decrement schemes for the GRAD pivot function. Once more, DPV was superior to the all other previously reported algorithms [79] [81] [164].



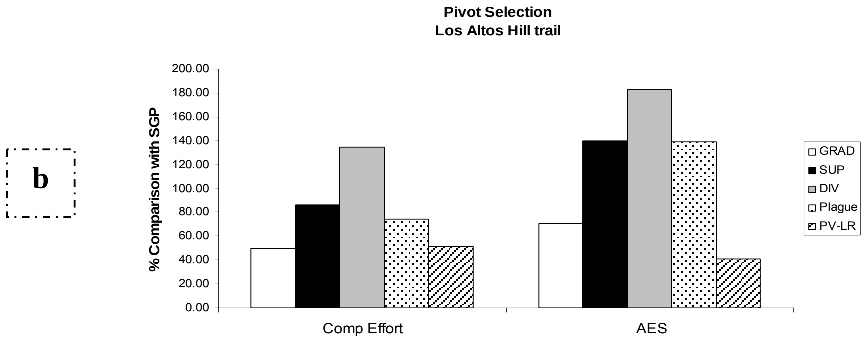


Figure 7.9 Pivot Selection Function – Santa Fe and Los Altos Hill trail

7.4 Conclusion

In this chapter the concept of DPV was introduced. The Dynamic Population Variation used the various characteristic measures for pivot functions together with a newly defined gradient based pivot function and a proportionate population increment/decrement scheme. The focus of this chapter was to investigate whether these newly introduced ideas can in any way have significant impact on GP performance in terms of computational effort. The main aim was to save computational effort by the dynamic adaptation of population size, that is by varying the population size by either incrementing or decrementing during the GP run using a heuristic feedback mechanism. These newly introduced schemes were experimented on three different problems on two separate landscapes for each problem to provide a good range of problem diversity. Experimental evidence has been produced to show that DPV can significantly improve performance by reducing the computational effort to reach a given fitness level. From this study, it can be concluded that the optimal pivot function is highly dependant on the amount of population variation, the population size, the fitness landscape and the pivot function's selected characteristic measure. For every specific problem, an optimal pivot function should exist which will depend on the actual characteristic measure used to determine stagnation. Consequently, every specific problem has its own most favourable combination of pivot function, population variation and characteristic measure. Although, there were some reductions in performance for some of the problems when the constant dependent population modification equation (7.4) was replaced with the generalised proportionate population increment/decrement equation (7.14), in most cases the performances for the problems studied were comparable. In fact, for some of the problems the generalised equation showed superior performance. The best performing schemes studied were the DIV pivot function using the HP mean characteristic measure with proportionate population increment/decrement for the degree-4 symbolic regression problem; the DIV pivot function using the HP mean characteristic measure with standard population increment/decrement for the degree-3 regression problem and simple induction problem; the GRAD pivot function using the HP mean characteristic measure with standard population increment/decrement for the parity problem; the GRAD pivot function with upper quartile selection and proportionate population increment/decrement for the Boolean symmetry problem; and the SUP pivot function with upper quartile selection and standard population increment/decrement for the recursive sequence induction problem. It was also discovered that the high performing mean and upper quartile were more suitable characteristic measures to use than the single best performing individual for the problems considered in this study. The proposed innovations [165] were shown to be superior to SGP, the plague operator [79] and all the static PV schemes previously reported in [164], at least for the representative problems studied here.

CHAPTER 8

EVALUATION AND ANALYSIS

"An expert is a man who has made all the mistakes, which can be made, in a very narrow field",
Niels Bohr.

8.1 A Statistical Analysis

In the previous sections, the workings and the superiority in performance of the static Population Variation (PV) and Dynamic Population Variation (DPV) was demonstrated. In this subsection, statistical inference is used to verify their superiority with some level of confidence across the various problems studied herein. A sample statistic such as $\overline{X}$ (sample mean) can vary from sample to sample and by taking into consideration this variability an interval estimate of the population mean can be developed. This interval can be used in estimating the population mean with a specified confidence. The level of confidence is symbolised by $(1-\alpha)\times 100\%$, where α is the proportion in the tails of the distribution which is outside the confidence interval. In other words, the proportion in the upper tail of the distribution is $\alpha/2$ and the proportion in the lower tail of the distribution is also $\alpha/2$. For example, an α of 0.05 corresponds to a 95% confidence. The confidence interval for a mean of a population μ with an unknown population standard deviation σ can be estimated for a sample of size n by using (8.1) and a t-distribution with n-1 degrees of freedom for an area of $\alpha/2$ in the upper tail. $\overline{X}$ is the sample mean and S is the sample standard deviation.

$$\overline{X} - t_{n-1}\frac{S}{\sqrt{n}} \leq \mu \leq \overline{X} + t_{n-1}\frac{S}{\sqrt{n}} \qquad (8.1)$$

Figure 8.1 shows the distribution of the computational effort for SGP and PV for the degree-4 regression problem. The distribution in the solid darker line represents the PV scheme and the distribution in the lighter line represents the SGP. The confidence intervals for each of the algorithms are computed using (8.1) with a 95% level of confidence. It can be said with 95% confidence that the population mean is estimated to be between 11825.8 and 28508.2 for the PV scheme and between 30109.3 and 53791.7 for the SGP. Appendix G contains the critical points of the t-Distribution.

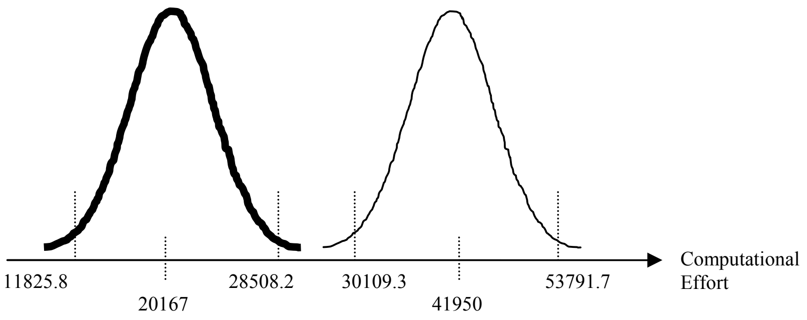


Figure 8.1 Distribution of Computational Effort for SGP and PV for the degree-4 regression problem

The above discussion provides the necessary background and foundation for the inferential statistical formulation that will be used in this subsection for comparing the performance means of various schemes. The formal or more specifically the standard methodology for comparing statistics from two samples of data (or schemes) drawn from two populations involves the "two-sample test" in conjunction with the hypothesis-testing methodology. Hypothesis testing begins with some claim or assertion about a particular parameter of a population. The null hypothesis (H_0) refers to the hypothesis that the population parameter is equal to a certain specification. The specification of a null hypothesis is generally combined with the specification of the alternative hypothesis (H_1) (or sometimes referred to as the research hypothesis). The alternative hypothesis is the opposite of the null hypothesis (H_0). The hypothesis-testing methodology is designed such that the rejection of H_0 is based on evidence from the sample data that the alternative hypothesis (H_1) is far more likely to be true. This methodology provides clear definitions for evaluating differences.

The test statistics used to compare statistics from samples of data drawn from two populations is the t-Test for the difference between two means. This test statistic is based on the difference between the sample means, i.e. $(\overline{X}_1 - \overline{X}_2)$, where $\overline{X}_1$ denotes the mean of the sample taken from population 1 and $\overline{X}_2$ denotes the mean of the sample taken from population 2. The sample size taken from the first and second population are denoted by n_1 and n_2 respectively. Similarly, μ_1 and μ_2 represent the respective population means from each population and σ_1 and σ_2 represent the population standard deviations from each population. In most cases, the variances and the standard deviations of the two populations are not known and the only information available is the sample means ($\overline{X}_1$ and $\overline{X}_2$), the sample variances or the sample standard deviations (S_1 and S_2). In this instance, the t-Test for the difference between two means is used to verify whether there is a significant difference between the means of the two populations. This statistic follows a t-Distribution with n_1+n_2-2 degrees of freedom. For a specified level of significance α, the null hypothesis is rejected if the computed t-test statistic is greater than the critical value from the t-Distribution. Appendix G contains the critical values from the t-Distribution. The t-Test statistic can be computed using (8.2) and (8.3), where S_p^2 denotes the pooled variance.

$$t = \frac{(\overline{X}_1 - \overline{X}_2) - (\mu_1 - \mu_2)}{\sqrt{S_p^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} \qquad (8.2)$$

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{(n_1 - 1) + (n_2 - 1)} \qquad (8.3)$$

The above mentioned hypothesis-testing methodology together with the t-Test for the difference between two means is utilised in this section to evaluate the performance means of the static population variation and dynamic population variation for the various problems detailed in this thesis.

The following null hypothesis and alternative hypothesis are formulated as per (8.4) for the evaluation of the performance means. The desired outcome is the rejection of the null hypothesis implying that there is a difference between the performance means among the two different schemes and the differences are not a resultant of chance and usage of one scheme over the other will provide the better performance.

$$H_0: \quad \mu_1 = \mu_2 \qquad (8.4)$$

$$H_1: \quad \mu_1 \neq \mu_2$$

Table 8.1 shows the summary of the statistical computation for the degree 4 Regression Problem for the static population variation. The critical value is obtained from the t-Distribution for a degree of freedom of 38 and level of significance of 0.05. This corresponds to a 95% level of confidence. The t-Test value is calculated using (8.2) and (8.3). As the computed t-Test is larger than the critical value obtained from the t-Distribution tables, the null hypothesis is rejected and the alternative hypothesis (H_1) is far more likely to be true. This means that the performance means of the two algorithms is not the same and there is a significant difference in performance between the two algorithms. In this instance, the PV scheme shows its superiority as expected.

Regression Problem – Degree 4		
	SGP	**PV**
Mean	41950	20167
Standard Deviation	17889.94	12602.55
t-Test	4.4517	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.1 Summary of the statistical computation for the degree 4 Regression Problem (PV Scheme)

Table 8.2 shows the summary of the statistical computation for the degree 4 Regression Problem for the dynamic population variation. This table indicates that there is sufficient evidence to reject the null hypothesis (H_0). It can be concluded that the performance means of the two schemes are different. Based on these results, the computational burden on GP using dynamic population variation appears to be lower than that of the SGP.

Regression Problem – Degree 4		
	SGP	DPV
Mean	41950	14871
Standard Deviation	17889.94	5070.67
t-Test	6.5126	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	Reject H_0	

Table 8.2 Summary of the statistical computation for the degree 4 Regression Problem (DPV Scheme)

Table 8.3 to Table 8.16 provide the summary of the statistical computation for each of the remaining problems on the two different landscapes. Similar conclusions and comments as per above apply to these problems.

Regression Problem – Degree 3		
	SGP	PV
Mean	3800	2996
Standard Deviation	547.72	637.16
t-Test	4.2783	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	Reject H_0	

Table 8.3 Summary of the statistical computation for the degree 3 Regression Problem (PV Scheme)

Regression Problem – Degree 3		
	SGP	**DPV**
Mean	3800	1606
Standard Deviation	547.72	772.96
t-Test	10.3554	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.4 Summary of the statistical computation for the degree 3 Regression Problem (DPV Scheme)

Digital Problem–Parity Checker		
	SGP	**PV**
Mean	237857	206471
Standard Deviation	25497.30	22665.39
t-Test	4.1144	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.5 Summary of the statistical computation for the Digital Parity Checker Problem (PV Scheme)

Digital Problem–Parity Checker		
	SGP	**DPV**
Mean	237857	189120
Standard Deviation	25497.30	10788.29
t-Test	7.8727	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.6 Summary of the statistical computation for the Digital Parity Checker Problem (DPV Scheme)

Digital Problem–Bool. Symmetry		
	SGP	**PV**
Mean	146000	105120
Standard Deviation	50225.81	36229.39
t-Test	2.9521	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.7 Summary of the statistical computation for the Digital Boolean Symmetry Problem (PV Scheme)

Digital Problem–Bool. Symmetry		
	SGP	**DPV**
Mean	146000	75134
Standard Deviation	50225.81	32923.37
t-Test	5.2772	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.8 Summary of the statistical computation for the Digital Boolean Symmetry Problem (DPV Scheme)

Sequence Induction–Simple		
	SGP	**PV**
Mean	52080	15660
Standard Deviation	31808.4	13346.33
t-Test	4.7217	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.9 Summary of the statistical computation for the Simple Sequence Induction Problem (PV Scheme)

Sequence Induction–Simple		
	SGP	DPV
Mean	52080	9303
Standard Deviation	31808.4	12791.35
t-Test	5.5800	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	Reject H_0	

Table 8.10 Summary of the statistical computation for the Simple Sequence Induction Problem (DPV Scheme)

Sequence Induction–Fibonacci		
	SGP	PV
Mean	6547500	2937087
Standard Deviation	1676770.47	455915.61
t-Test	9.2920	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	Reject H_0	

Table 8.11 Summary of the statistical computation for Fibonacci Sequence Induction Problem (PV Scheme)

Sequence Induction–Fibonacci		
	SGP	DPV
Mean	6547500	1266580
Standard Deviation	1676770.47	675721.69
t-Test	13.0639	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.12 Summary of the statistical computation for Fibonacci Sequence Induction Problem (DPV Scheme)

Artificial Ant–Santa Fe		
	SGP	PV
Mean	42000	24240
Standard Deviation	41091.68	16983.53
t-Test	1.7863	
Level of Confidence	90%	
t_{38} (from t-Distribution)	1.686	
Decision	**Reject H_0**	

Table 8.13 Summary of the statistical computation for Artificial Ant Problem on Santa Fe trail (PV Scheme)

Artificial Ant–Santa Fe		
	SGP	**DPV**
Mean	42000	5309
Standard Deviation	41091.68	4838.94
t-Test	3.9658	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.14 Summary of the statistical computation for Artificial Ant Problem on Santa Fe trail (DPV Scheme)

Artificial Ant–Los Altos Hills		
	SGP	**PV**
Mean	16600	7636
Standard Deviation	6199.66	3139.43
t-Test	5.7687	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.15 Summary of the statistical computation for Artificial Ant Problem on Los Altos Hills trail (PV Scheme)

Artificial Ant–Los Altos Hills		
	SGP	**DPV**
Mean	16600	8243
Standard Deviation	6199.66	3939.34
t-Test	5.0880	
Level of Confidence	95%	
t_{38} (from t-Distribution)	2.0244	
Decision	**Reject H_0**	

Table 8.16 Summary of the statistical computation for Artificial Ant Problem on Los Altos Hills trail (DPV Scheme)

For each of the above tables the t-Test values are calculated as per (8.2) and (8.3). Once the level of significance is set, the critical values for the appropriate statistical distribution can be found. It should be pointed out that the level of confidence was 95%, except for one of the problems (Artificial Ant – Santa Fe trail) where it was 90%. The critical values divide the rejection and non-rejection regions as shown in Figure 8.2. The computed test statistics are compared with the critical value to determine whether the test statistic falls within the rejection or non-rejection region. The statistical decision of whether the null hypothesis (H_0) is rejected is based on this comparison of the computed test statistic with the critical value. If the computed test statistic is greater than the critical value from the t-Distribution, i.e. if it falls within the rejection region, then the null hypothesis (H_0) is rejected. By rejecting the null hypothesis (H_0) at a given level of significance α, it is statistically proven that the alternative hypothesis (H_1) is correct. This would indicate that there is a difference in the performance means and based on the results the proposed population variation schemes do show their superiority.

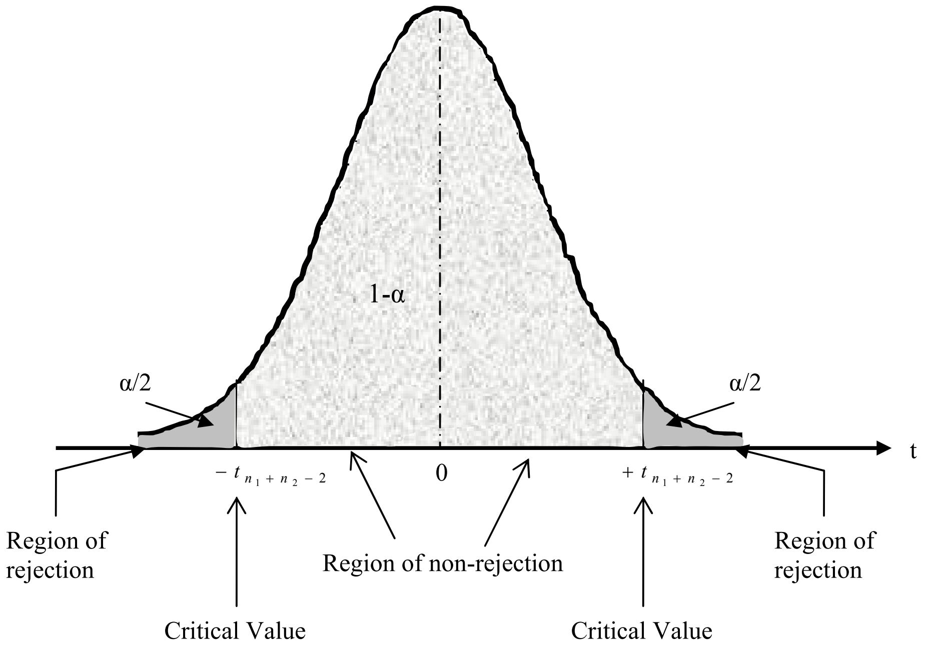


Figure 8.2 Regions of rejection and non-rejection for the t-Test for the difference between two means

8.2 A Preliminary Word on Diversity

Maintaining or increasing diversity is generally and conventionally considered as beneficial in GP. The importance of diversity as a crucial feature in avoidance of premature convergence is consistently cited in the GP literature [73] [223] [227] [273] [275]. There exist numerous possible definitions of diversity in GP. The term diversity can be referred to as the diversity of genotypes (structural diversity) or behavioural difference (phenotypes). Structural differences do not guarantee behavioural differences and frequently imply that two structures are not identical. The term variety was used in [170] to indicate structurally unique individuals or programs. Two identical structures will produce the same behaviour. Hence, it was argued [189] that genotypic diversity is a sufficient upper bound of population diversity. Consequently, a decrease in genotypic diversity would necessarily cause a decrease in unique fitness responses. The structural representation in GP lends itself to more fine grain structural measures. Thus, many structural diversities have been defined such as the number of different structures (individuals, programs, or genotypes) [189], the edit distance between structures in the population [58] [73] and other composite measures [49] [151]. For example, edit distance diversity, which provides a fine grain description of population structural differences, is based on the distance between every individual in the population and the best fit individual found so far in the run. In the standard edit distance measure two trees are overlapped at the root node counting the non-identical overlapping nodes. Two different nodes score a distance of 1, whereas equal nodes score a distance of 0. The edit distance is then the sum of all different nodes which is normalised by dividing it by the size of the smaller tree. Another example providing a more abstract view of the population is the pseudo-isomorphs, which are less computationally expensive than edit distance and are found by defining a three-tuple of <terminals, non- terminals, depth> for each tree. The number of unique three-tuples is the diversity measure. The more detailed a measure is the more computation it will require and thus, it is required to find informative and inexpensive measures.

The measure of success is however the fitness of a solution or behaviour in the problem's environment. Such measures compare differences between the populations' fitness values at a given time. This is in line with the focus of GP which is driven by a performance goal or fitness improvement and not by the level of structural diversity. Moreover, the selection mechanism, which chooses individuals to produce the next generation, selects individuals based on fitness. Consequently, phenotypic diversity measures (measures based on fitness) are quite important. Some examples of phenotypic diversity measures are discussed in [272] [273]. It has been suggested that phenotypic diversity measures appear to be superior to genotypic measures [32] [33] [151] [275]. Consequently, in this chapter only phenotypic measures have been used for evaluation and analysis of diversity.

8.3 Phenotypic Diversity Measures

Genetic heterogeneity, or diversity, is considered as one candidate for useful statistical measure in this chapter. Specifically as described in the previous subsection phenotypic diversity, which counts the number of unique fitness values in a population, plays a significant role in evaluation of diversity. The phenotypic diversity measures used in this chapter are mainly based on the concepts of entropy and standard deviation. The classical interpretation of entropy *H* comes from the second law of thermodynamics introduced by Clausius to represent the change of state when an increment of energy (*dQ*) is added to a body as heat at an absolute temperature *T* during a reversible process as defined by (*8*.5).

$$\Delta H = \int_{\textit{initial state}}^{\textit{final state}} \left(\frac{dQ}{T} \right) \qquad (8.5)$$

This was later interpreted statistically by Boltzman. Entropy represents the disorder in the system of particles. Within the context of GP, the entropy measure represents the chaos of the system with respect to the distribution of fitness values. Entropy *H*(*P*) represents a measure of population diversity. The entropy describes the number of unique phenotypes and how the population is distributed over the existing phenotypes. An entropy measure was introduced in [273] as defined by (*8*.6):

$$H(P) = -\sum_{j} p_j \log_{10}(p_j) \qquad (8.6)$$

The fitness class p_j is the proportion of the population occupied by population partition. A partition is considered as each possible different fitness value, but could be defined to include a subset of values. High entropy describes the presence of many unique fitness values where low entropy describes a population which contains fewer unique fitness values and where many individuals may have the same fitness. Hence, an increase in entropy represents an increase in diversity. Although the entropy is consistently and mainly used as a measure for phenotypic diversity in the literature, in addition the standard deviation s is in some instances used in this chapter as defined by (*8*.7).

$$s(P) = \sqrt{\frac{1}{n-1}[\sum_{k=1}^{n}(f_k - \widehat{f})^2]} \qquad (8.7)$$

The fitness of each individual and the average fitness in the population of n individuals are denoted by f_k and $\widehat{f}$ respectively. Entropy plots and other statistics such as standard variation will be used in the following sections as a measure for phenotypic diversity.

8.4 Analysis of Static Population Variation

In this section, our understanding of static population variation is further extended by analysing and evaluating both the reduction and increment schemes. Specific data from multiple runs of specific problems are extracted to conduct this analysis.

8.4.1 Reduction Profiles

Three specific reduction profiles, namely linear, quadratic and exponential, as shown in Figure 8.3 are used for this analysis. Based on the reduction profiles, it is expected that PV-LR and PV-QR perform similarly with PV-ER behaving quite differently. Towards the initial stages of the run, the amount of reduction in population is in the order of PV-QR, PV-LR and PV-ER. However, towards the end of the run the order of individuals in the population is reversed. The question arises at which stage during the evolutionary process the presence of higher individuals is critical. It was noted in chapter 6 that the actual profile in use is in fact problem dependant. In addition, it was observed that as the amount of reduction was increased, there was an improvement in terms of computational effort. However, the improvement stopped once the amount of reduction was beyond a certain threshold. Any reduction beyond this value was considered to be detrimental as the GP did not have a sample space big enough to perform efficiently.

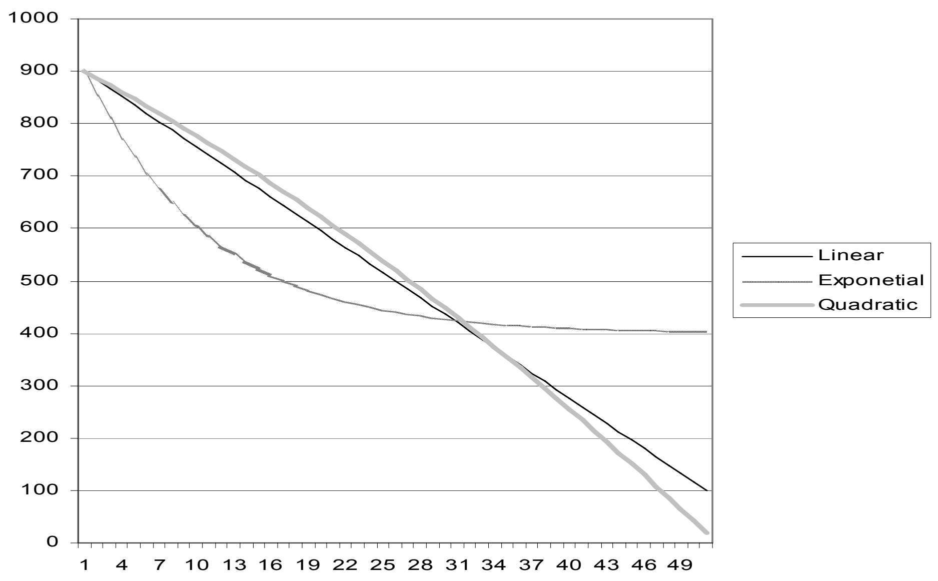


Figure 8.3 Sample linear, exponential and quadratic reduction profiles

As can be seen in Figure 8.4 and as it was expected based on the reduction profile, the PV-LR and PV-QR algorithms to perform similarly. Both of these algorithms exceeded the PV-ER scheme in terms of performance. Furthermore, there is a slight improvement in performance in PV-LR over PV-QR. This can clearly be observed in Figure 8.5 where it takes much less effort to reach a given fitness threshold for PV-LR than the other reduction profiles.

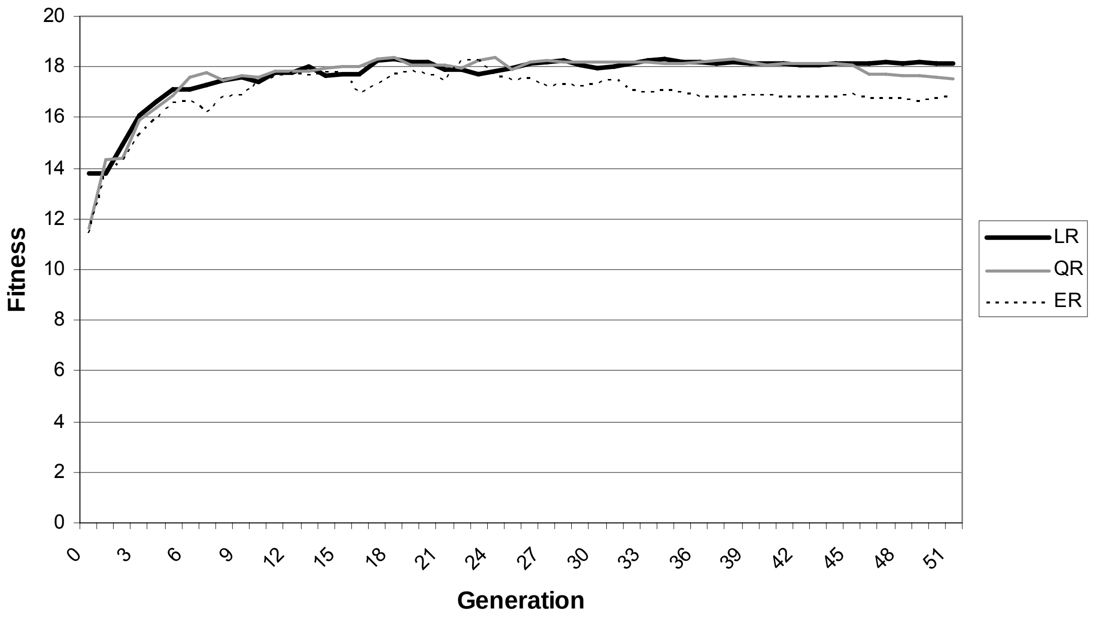


Figure 8.4 Reduction profiles Deg-4 Regression Problem: average best fitness-generation comparison

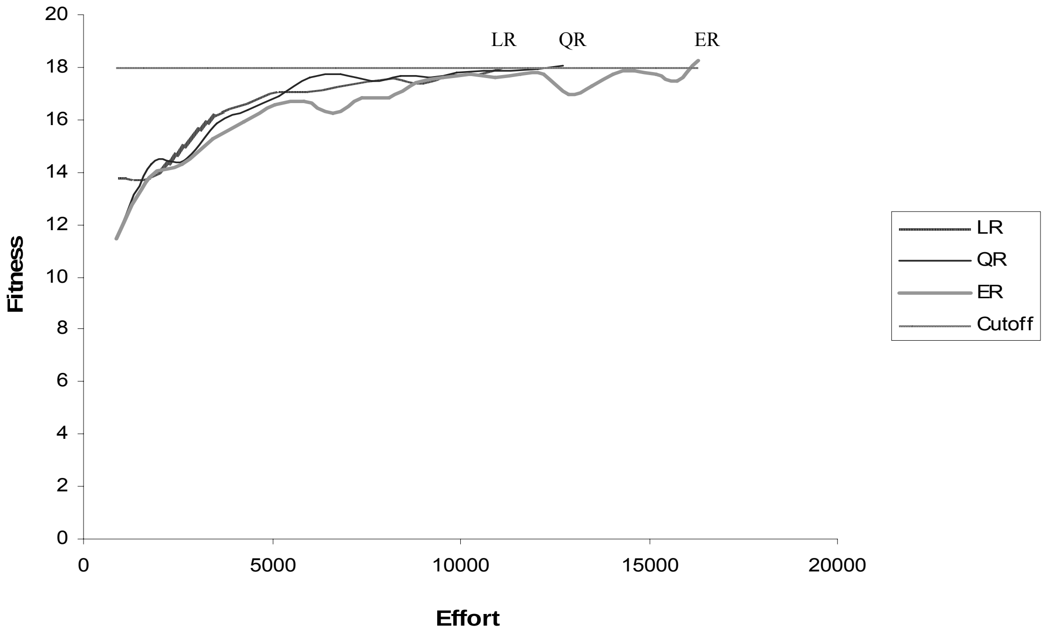


Figure 8.5 Reduction profiles Deg-4 Regression Problem –Effort for a given cut-off fitness

It is believed that PV-ER may suffer in performance due to lower population diversity. A quick glance at Figure 8.6 confirms this notion.

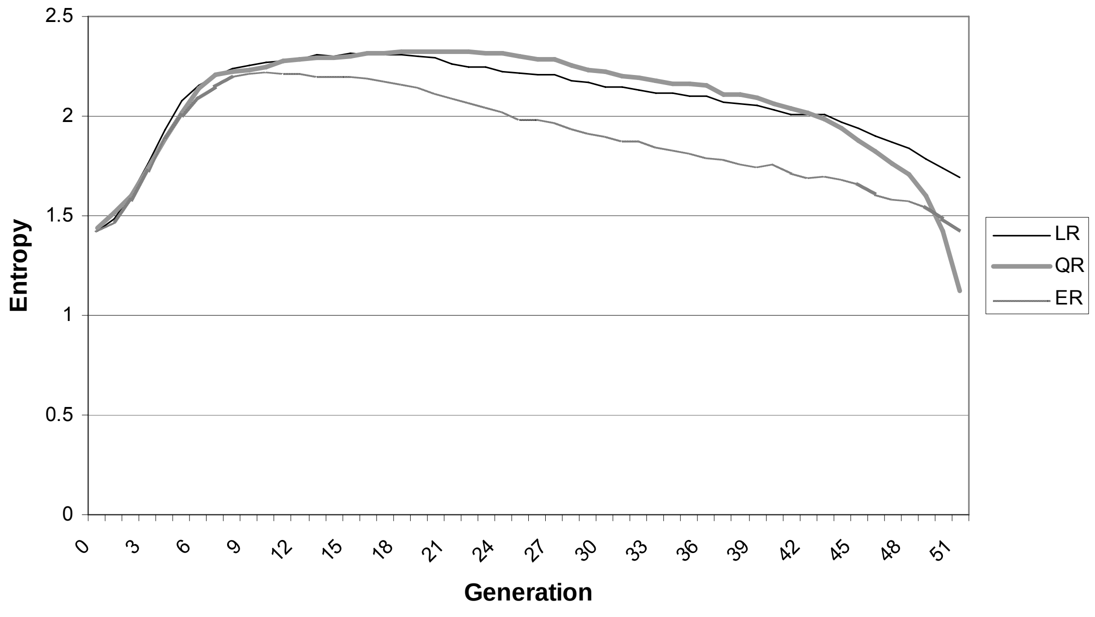


Figure 8.6 Reduction Profile Deg-4 Regression Problem - Entropy

Although the phenotypic diversity of PV-QR is more than PV-LR in the initial and middle stages of the evolutionary process, the extra computational burden is not worthwhile in discovering the global optimum. In this instance, the PV-LR seems to be the optimal reduction scheme. Any reduction scheme in the vicinity of this scheme or traversing similar reduction profile as that of the PV-LR profile will result in optimum performance and can be considered as the “threshold” for reduction. Any lesser reduction will perform worse than the threshold or optimum profile as it is subjected to higher computational effort due to higher population levels and superfluous burden in evaluating fitness values for extra individuals than are essential. Any higher reduction than this threshold will also be outperformed by the optimum profile, as it will suffer from the lack of indispensable population diversity to solve a given problem.

8.4.2 Increment Profiles

Three specific increment profiles of linear, quadratic and exponential are drawn on for the analysis of PV-I, as shown in Figure 8.7. Based on the profiles, it is expected that PV-LI and PV-QI achieve similar results as far as computational effort to reach an expected fitness is concerned, but it is expected that PV-EI will act a bit differently. PV-EI starts off with a larger value of initial population whereas PV-LI and PV-QI begin with a very small initial population.

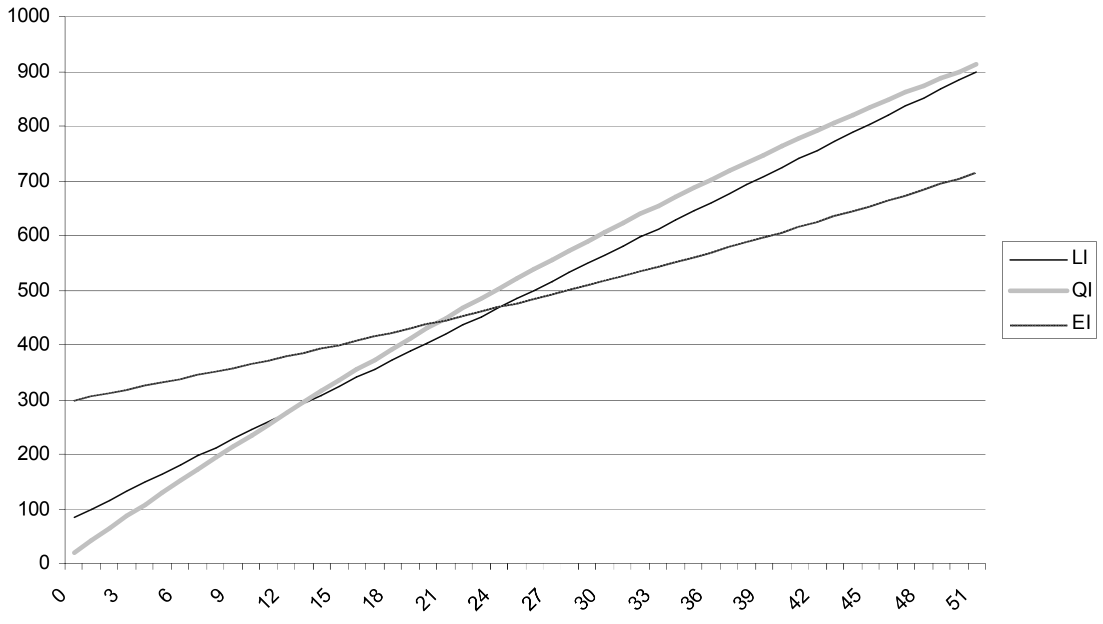


Figure 8.7 Sample linear, exponential and quadratic increment profiles

The amount of increment for PV-EI is steady whereas for PV-QI and PV-LI it is much larger. Moreover, the final population size of the PV-LI and PV-QI is a good deal larger than the PV-EI scheme. The main question is if the evolutionary process starts with a much smaller population size, does it still have the capacity to progress productively. In this instance, the GP will be well ahead than for the other case when it starts off with a much larger population size, as it does not unnecessarily waste computational effort at the beginning, where the majority of the individuals are performing poorly, and can use its main computational effort later on when the individuals have evolved and may have a higher fitness value. Another matter that may require consideration is at which stage during the evolutionary process that having a larger number of individuals can result in a better performance. This would determine the incremental value of the population size and the timing within the evolutionary cycle. The best performing scheme in terms of computational effort, for this problem, is the PV-LI, as can be seen in Figure 8.8.

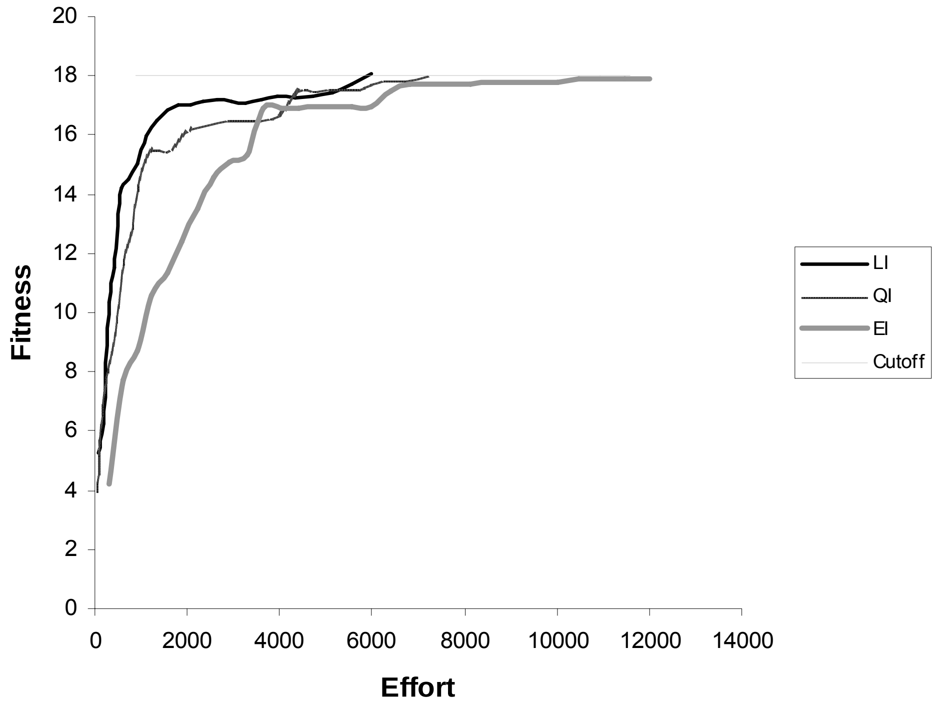


Figure 8.8 Increment profiles Deg-4 Regression Problem –Effort for a given cut-off fitness

This highlights the fact that a large population size after the individuals have evolved, containing high fitness values, has a much better impact for this problem than a large population size initially where almost all the individuals are performing poorly. PV-LI outperforms PV-QI indicating that the extra individuals at the beginning were critical to its success, implying that there exists some critical population size that GP will need to start off with. This should be reflected in the fitness plot (Figure 8.9), where PV-LI should have individuals with much higher fitness at the beginning. One would also expect that the entropy (Figure 8.10) to be higher for the PV-LI than for PV-QI. In addition, the entropy should be an incrementing function as new individual are continuously injected into the population.

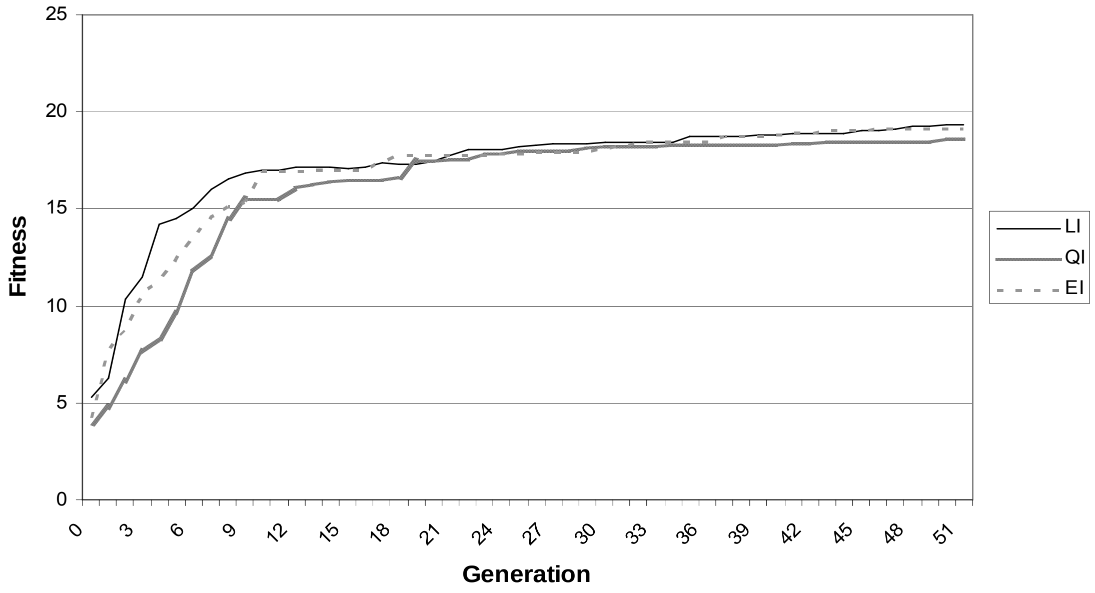


Figure 8.9 Increment profiles Deg-4 Regression Problem: average best fitness-generation comparison

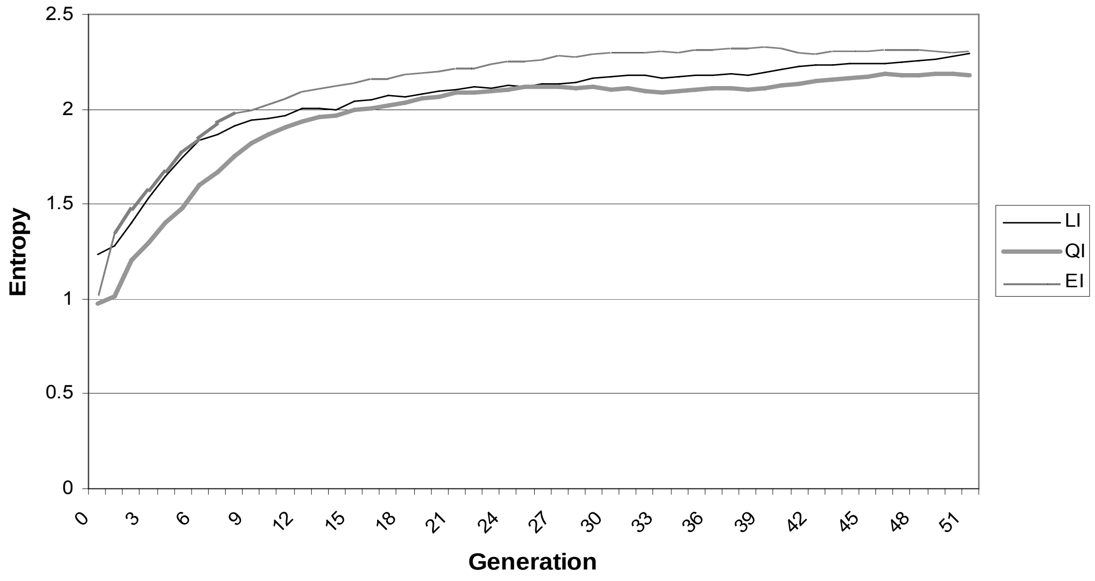


Figure 8.10 Increment Profile Deg-4 Regression Problem – Entropy

8.4.3 Increment versus Decrement

In this section, the best performing increment scheme for this problem, namely PV-LI is weighed against the best performing decrement scheme PV-LR. As per the previous subsection, when different increment schemes were being compared, it was observed that a higher starting population would not be as effective as starting with a lower population size such as in the PV-LI. Accordingly, it is expected that PV-LI should outperform PV-LR, which can clearly be seen in Figure 8.11. PV-LR starts with much higher fitness (Figure 8.11 and Figure 8.12) as it has more individuals at the initial stage of evolution but soon PV-LI catches up, having expended much less computational effort.

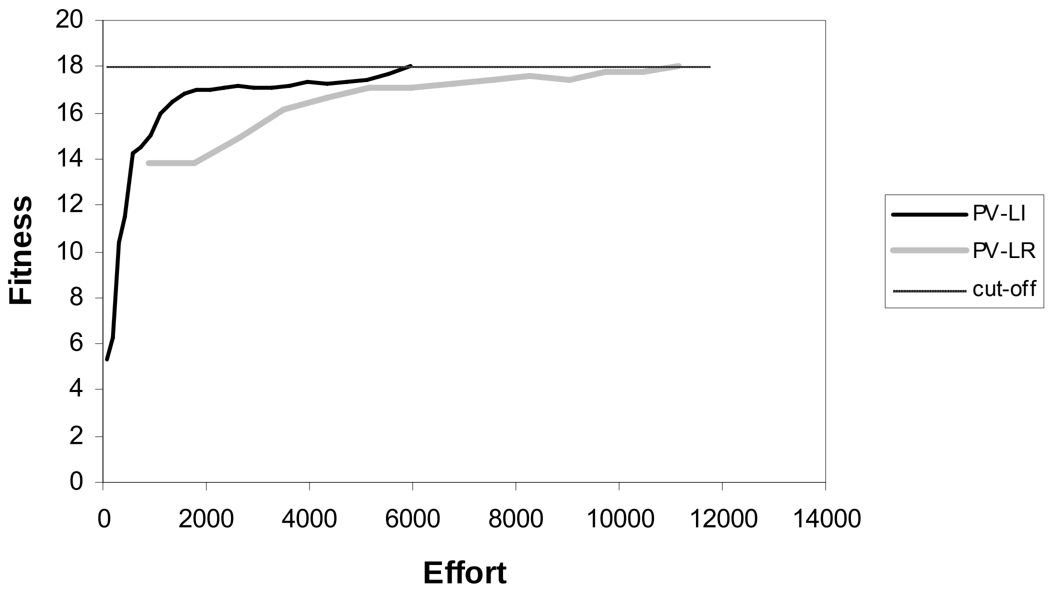


Figure 8.11 PV-LI vs. PV-LR - Deg-4 Regression Problem (Effort-Fitness)

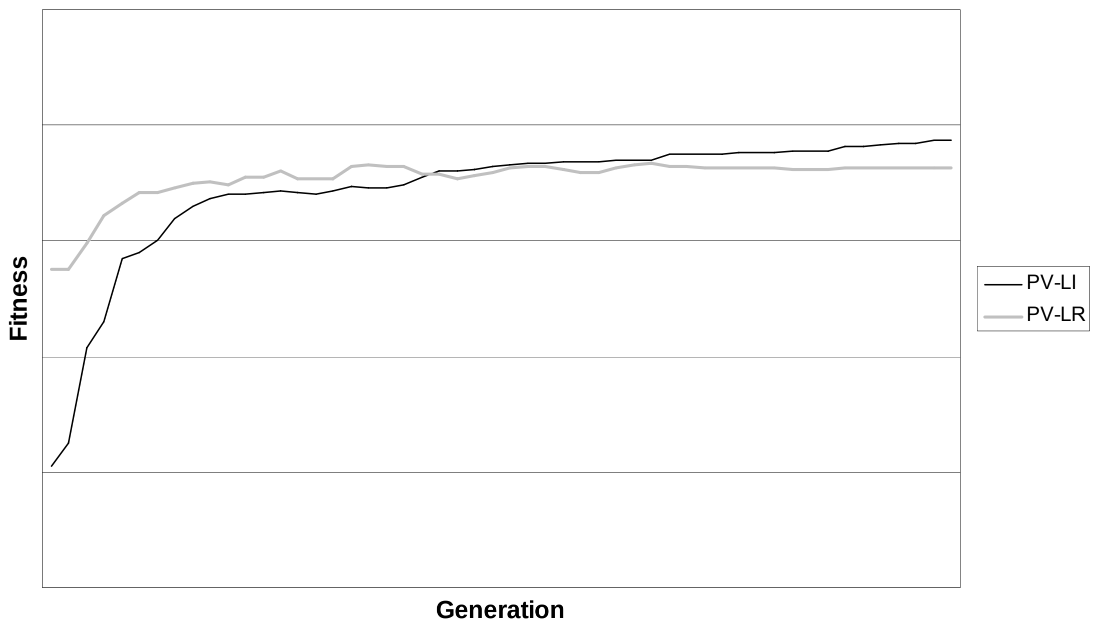


Figure 8.12 PV-LI vs. PV-LR - Deg-4 Regression Problem (Fitness-Generation)

In addition, the fitness continues to improve in PV-LI as new individuals are continuously infused into the system, whereas in PV-LR fitness steadily approaches an asymptote. It is believed that this could be the reason for converging or decreasing diversity in the latter case while an increasing diversity in the former situation. A diversity plot can plainly reveal this idea and validate this hypothesis.

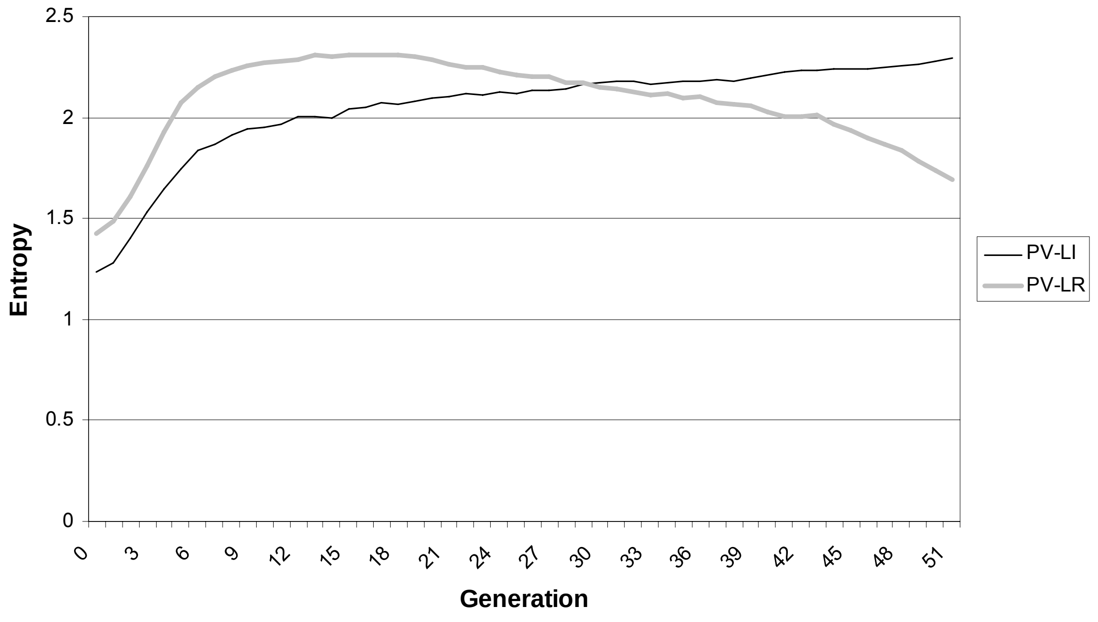


Figure 8.13 PV-LI vs. PV-LR- Deg-4 Regression Problem (Entropy)

As can be seen in Figure 8.13, in the case of PV-LR the diversity initially is larger than in the case of PV-LI, as it starts off with a much larger initial population size, and the diversity continues to grow as the GP starts exploring the landscape but as there is a continuous removal of individuals PV-LR eventually experiences a reduction in its phenotypic diversity. On the other hand, PV-LI continuously instils new individuals into the evolutionary process increasing the diversity. Although, the standard deviation of the two schemes follows similar trends, towards the end of the generation a significant reduction in this measure is detected (Figure 8.14).

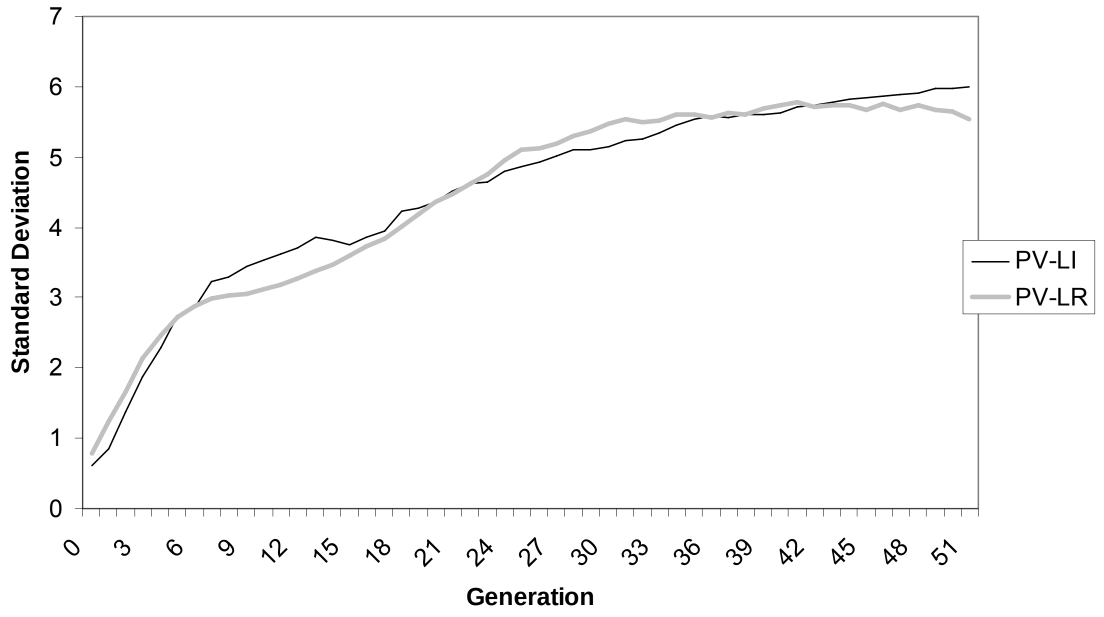


Figure 8.14 PV-LI vs. PV-LR- Deg-4 Regression Problem (Standard Deviation)

8.4.4 Increment Does Not Work in Some Instances

Two problems (Deg-4 Regression Problem and artificial ant problem), where PV-LI is successful and unsuccessful, are compared. It is believed that PV-I will not be successful if it starts off with a population size that is below the value of the critical size which is necessary for the individuals to evolve effectively. In this instance, it almost gets stuck at a given fitness level, as it did not start with a population level rich enough to progress. More importantly, for problems where there is not much room for improvement (when the fitness landscape is flat), PV-I cannot function efficiently. At these instances a reduced population level would be more beneficial. Figure 8.15 shows the incremental fitness improvement thus far for the two problems where PV-LI was successful and unsuccessful. In the former case the fitness improvement was of the order of 67%, whereas in the latter case the fitness improvement was a mere 7%. In addition, it is believed that in problems where PV-I is successfully progressing the phenotypic diversity will steadily increase, whereas in problems where PV-I is performing poorly, the diversity would decreasingly converge.

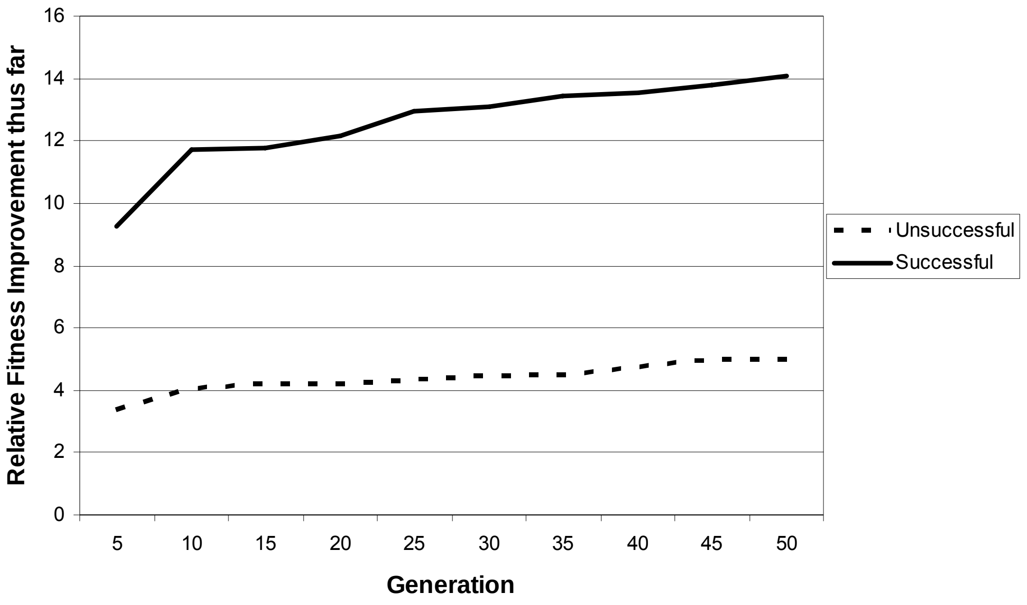


Figure 8.15 Incremental Fitness Improvement thus-far

Three measures of phenotypic diversity of range, standard deviation and entropy are used to illustrate this point. The Fitness Range is the difference between the *best-performing* and the *worst-performing* individuals thus far. As it can be seen in Figure 8.16 the Fitness Range is continually increasing in the problem where PV-LI is operating efficiently, whereas in the problem where it is not as effective, there are fluctuations in the fitness range with a decreasing cyclic pattern. Figure 8.17 shows the standard deviation. Similar trends are observed for the measure of standard deviation, where it is an increasing function when PV-LI is performing well. Finally, the entropy for the two problems is displayed in Figure 8.18. As it can be seen from all the figures, for the problems that PV-LI is not suitable the phenotypic diversity measures all are decreasing functions.

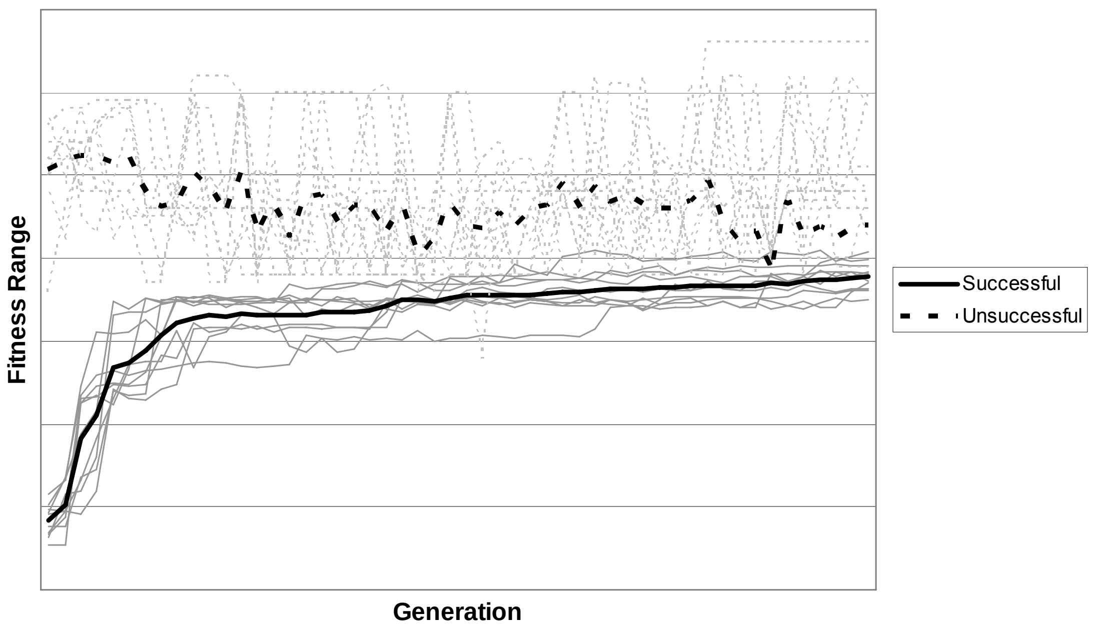


Figure 8.16 Fitness Range- Performance comparison of PV-LI in 2 problems

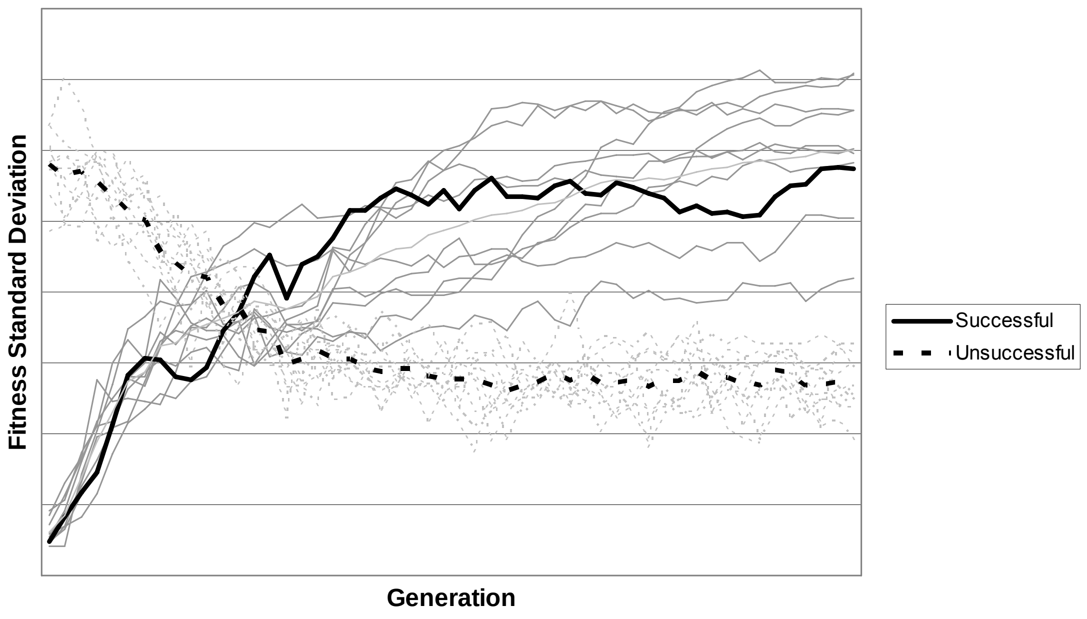


Figure 8.17 Fitness Std Deviation- Performance comparison of PV-LI in 2 problems

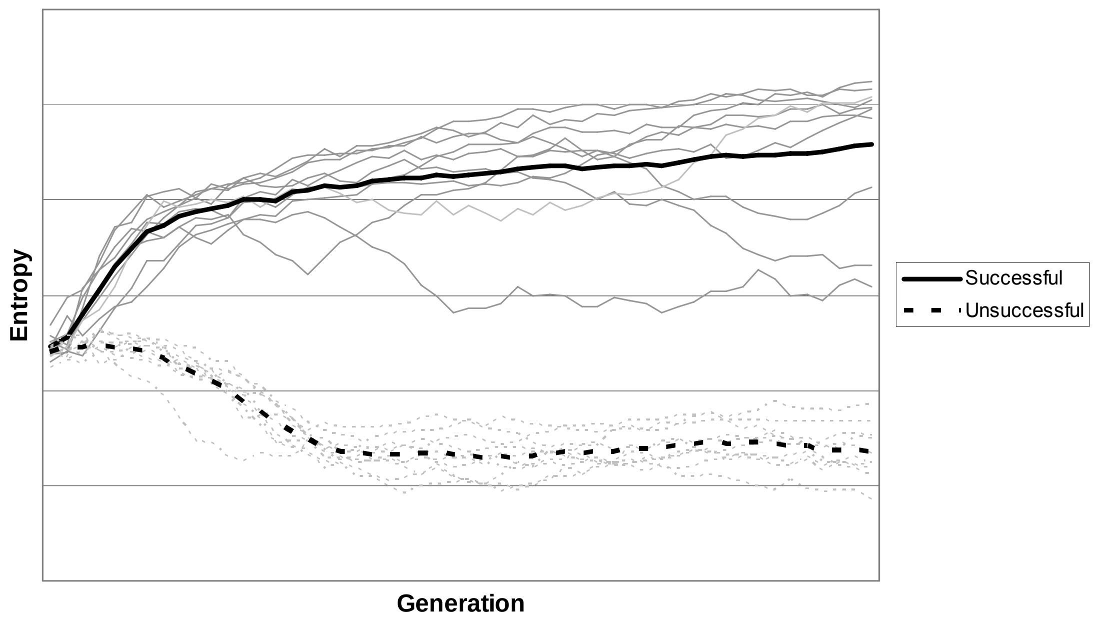


Figure 8.18 Entropy- Performance comparison of PV-LI in 2 problems

8.5 Dynamic Population Variation

By delving in at the pivot function, our understanding of dynamic population variation can be broadened. Next a brief examination of why Dynamic Population Variation works is undertaken. This section will finally conclude by taking a closer look into the stagnation phase. Explicit data from multiple runs of different problems are used to carry out this analysis.

8.5.1 A Closer Look at the Pivot Function

The population size is ultimately controlled by the pivot function. The pivot function determines whether population size is to be incremented or decremented. Table 8.17 can be deduced using equations 6.1, 6.2, 6.10, 6.11 and 6.13 with the period *T* set to 1. DEC denotes a reduction (decrement) in population size and INC an increment in population size. The main three columns in the table below are each of the three pivot functions DIFF, GRAD and DIV. The change in the characteristic measure from generation to generation could be very small or large. The population can be considered as progressing or regressing as far as the characteristic measure is concerned. However, the progress or regress from generation to generation may remain constant, improve or decline. P, R and N stand for progress, regress and No (or Nil) progress respectively. P-P identifies a progress in characteristic measure within the previous cycle followed with another observed progress in the existing cycle of the evolutionary process. Similarly, P-R defines a progress in characteristic measure in the previous cycle followed by an observed regress in the current cycle. Analogous definitions are used for P-N, N-P, N-N, N-R, R-P, R-N and R-R. This table is designated as "Pivot Function Table of Actions". The Table is divided into four main quadrants where there is an observed change in the characteristic measure. These four quadrants are highlighted with different levels of grey shading for easier identification and grouping of similar observed behaviour as far as the characteristic measure is concerned. The middle section of the table, which does not have any shading, signifies no change in the characteristic measure from generation to generation. Using the Pivot Function Table of Actions, the behaviour of each of the pivot functions (DIFF, GRAD and DIV) can easily be inferred.

		Small Changes			Large Changes		
		DIFF	GRAD	DIV	DIFF	GRAD	DIV
P-P	Improving	*DEC*	*DEC*	*DEC*	*DEC*	*DEC*	*DEC*
	Constant	*DEC*	*DEC*	*INC*	*DEC*	*DEC*	*DEC*
	Declining	*DEC*	*INC*	*INC*	*DEC*	*INC*	*DEC*
P-N	Stops	*DEC*	*INC*	*INC*	*DEC*	*INC*	*INC*
P-R	Regressing	*DEC*	*INC*	*DEC*	*DEC*	*INC*	*INC*
N-P	Progressing	*DEC*	*DEC*	*DEC*	*DEC*	*DEC*	*DEC*
N-N	No Progress	*DEC*	*DEC*	*INC*	*DEC*	*DEC*	*INC*
N-R	Regressing	*DEC*	*INC*	*INC*	*DEC*	*INC*	*INC*
R-P	Progressing	*INC*	*DEC*	*DEC*	*INC*	*DEC*	*DEC*
R-N	No Progress	*INC*	*DEC*	*INC*	*INC*	*DEC*	*INC*
R-R	Improving	*INC*	*DEC*	*INC*	*INC*	*DEC*	*INC*
	Constant	*INC*	*DEC*	*INC*	*INC*	*DEC*	*INC*
	Declining	*INC*	*INC*	*INC*	*INC*	*INC*	*INC*

Table 8.17 Pivot Function Table of Actions

When the characteristic measure is the *best-performing* individuals the behaviour of the pivot becomes more predictable. Firstly, as elitist is utilised we can only be in the middle section of the table and in the first and second quadrants, i.e. the top quadrants with the lightest grey shadings. Therefore, DIFF will constantly remove individuals from the population. This is clearly visualised in the Figure 8.19.

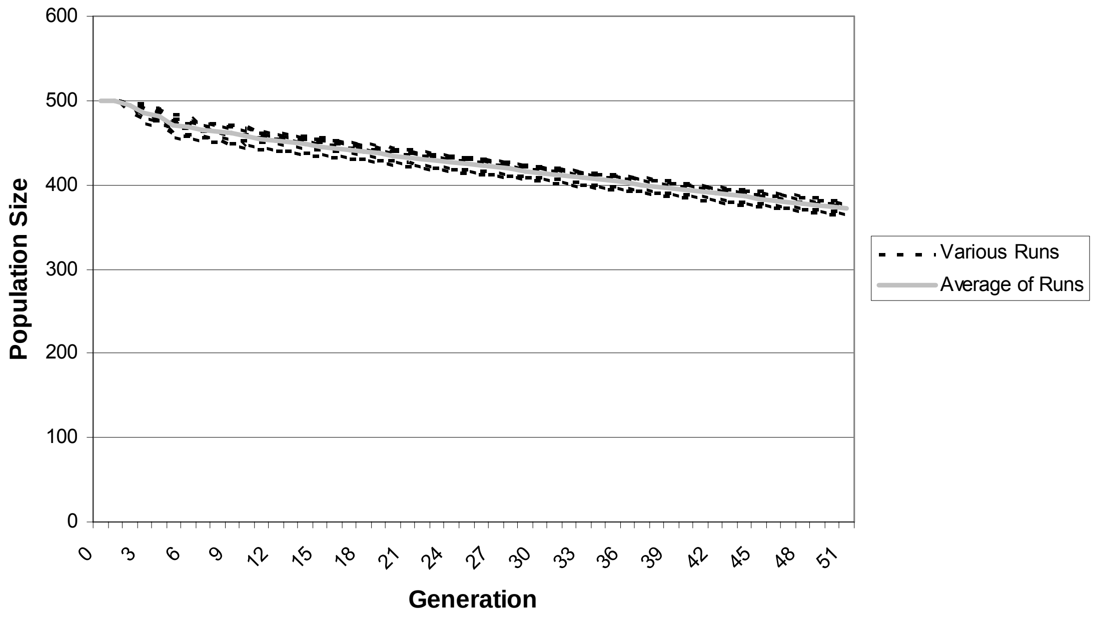


Figure 8.19 Population Size for DIFF pivot function using Elitist Strategy when characteristic measure is the *best-performing* individuals – Regression Problem (Deg-4)

Towards the beginning of the evolutionary cycle, it is expected that the fitness of the best-performing individuals improve, however as the evolutionary process progresses, it will be difficult to maintain this level of improvement and hence it is anticipated that the fitness values are either improving in small steps or the progress comes to a halt. Therefore, the drift of the population levels would be increasing for DIV and decreasing for DIFF. This tendency of decrease initially in the beginning followed by increase in population size for DIV is clearly discerned in Figure 8.20.

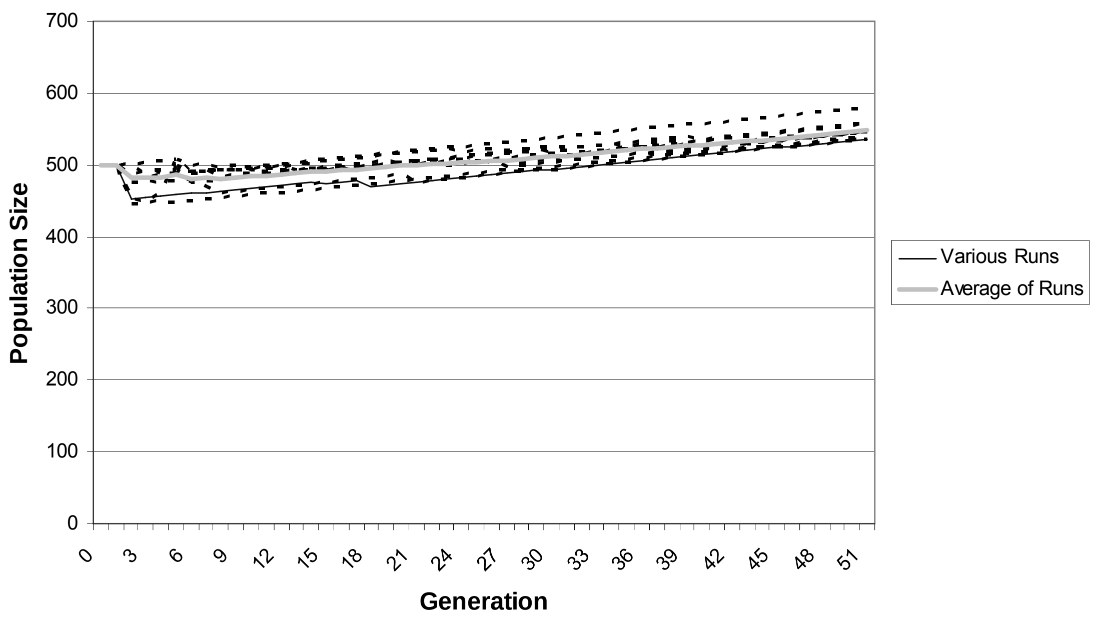


Figure 8.20 Population Size for DIV pivot function using Elitist Strategy when characteristic measure is the *best-performing* individuals - Deg-4 Regression Problem

It should be noted that such generalisations cannot be deduced for other characteristic measures, such as mean or average high performing individuals etc. (Figure 8.21 and Figure 8.22). Although, it can be predicated that the tendency for DIFF will be the removal of individuals for the mean characteristic measure, as the mean will generally be increased from generation to generation, it would not be a continually decreasing function as was the case in DIFF with *best-performing* individuals (Figure 8.19). In this instance, there will be stages where the population level will be increased (see Figure 8.21) and hence it would be estimated that the final population level for DIFF with mean selected as the characteristic measure would be higher than for DIFF with *best-performing* individuals selected as the characteristic measure. The location of these surges of increments would be quite unpredictable.

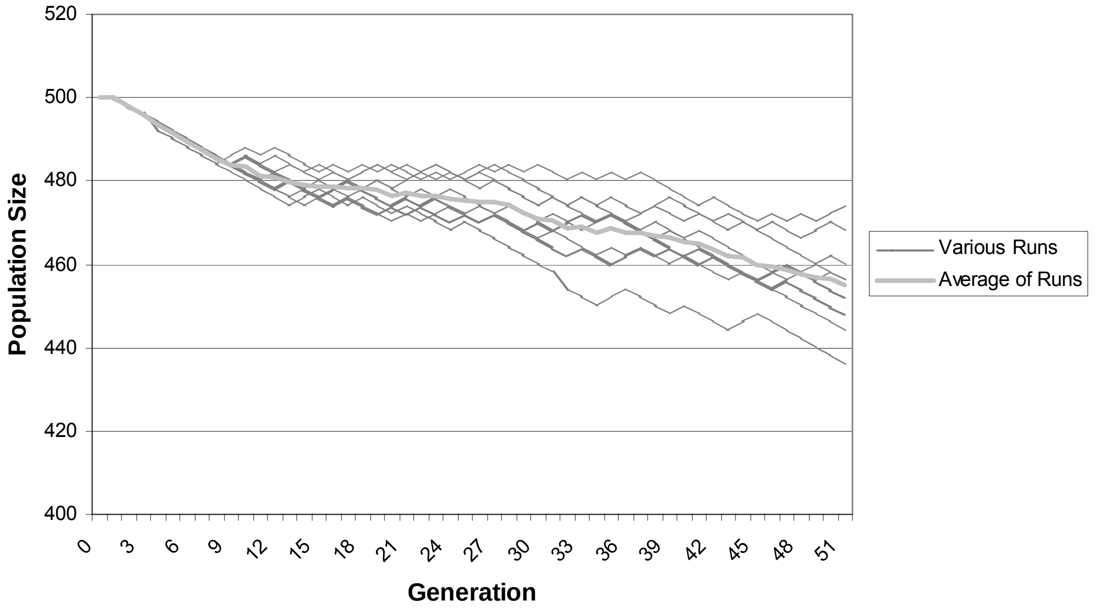


Figure 8.21 Population Size for DIFF pivot function using Elitist Strategy when characteristic measure is the mean fitness- Deg-4 Regression Problem

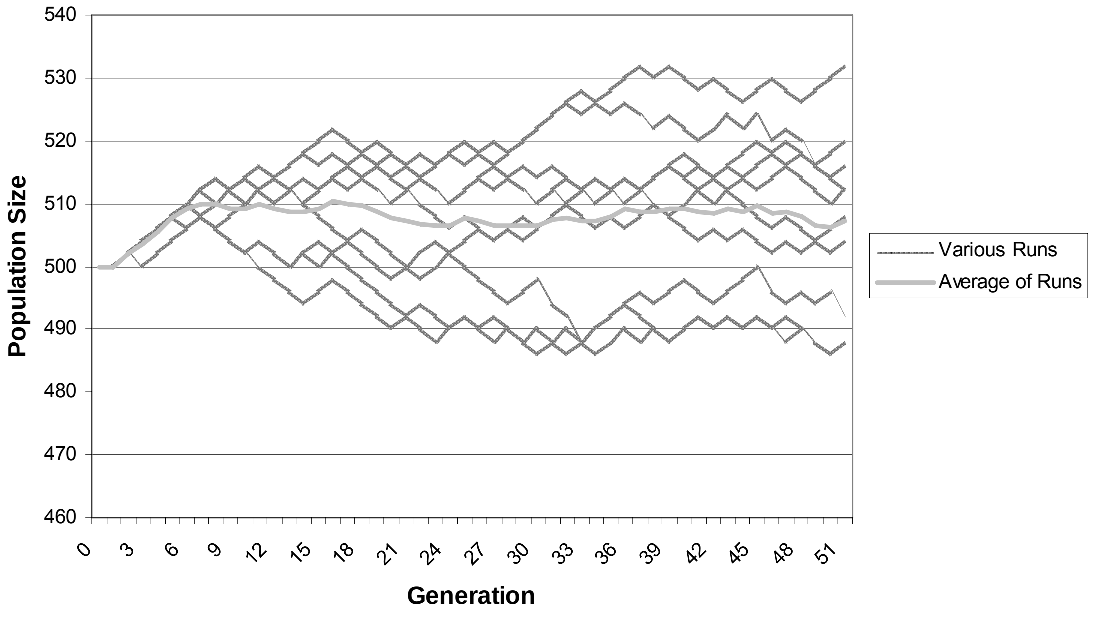


Figure 8.22 Population Size for DIV pivot function using Elitist Strategy when characteristic measure is the mean fitness- Deg-4 Regression Problem

The combination of pivot function and characteristic measure ultimately determines the reduction or increment of the population size and thereby indirectly controls the progress of the GP system within the evolutionary cycle.

8.5.2 Why DPV Works

Dynamic Population Variation provides an opportunity for the population size to grow and to diminish in the same run of an evolutionary cycle in response to specific measures with a heuristic feedback mechanism for decision making. These specific measures can be considered to be sensors of a desired criterion such as performance or diversity etc.

The working of Dynamic Population Variation can be best expounded by an example and the following explanation. While there is significant improvement observed with the characteristic measure, the Dynamic Population Variation simply removes individuals from the population set and thereby saves on computational effort, which may have been unnecessarily wasted. The extra individuals at this stage of the evolutionary process may be regarded as redundant. However, as the improvements come to stagnation or commence to decline, the Dynamic Population Variation increases the population size by injecting new individuals (fresh blood) to ensure that the improvements can be maintained and steadily continue. At this time, the increased level of population size is justified and does not result in a redundant computational effort. This concept is clearly exemplified in Figure 8.23.

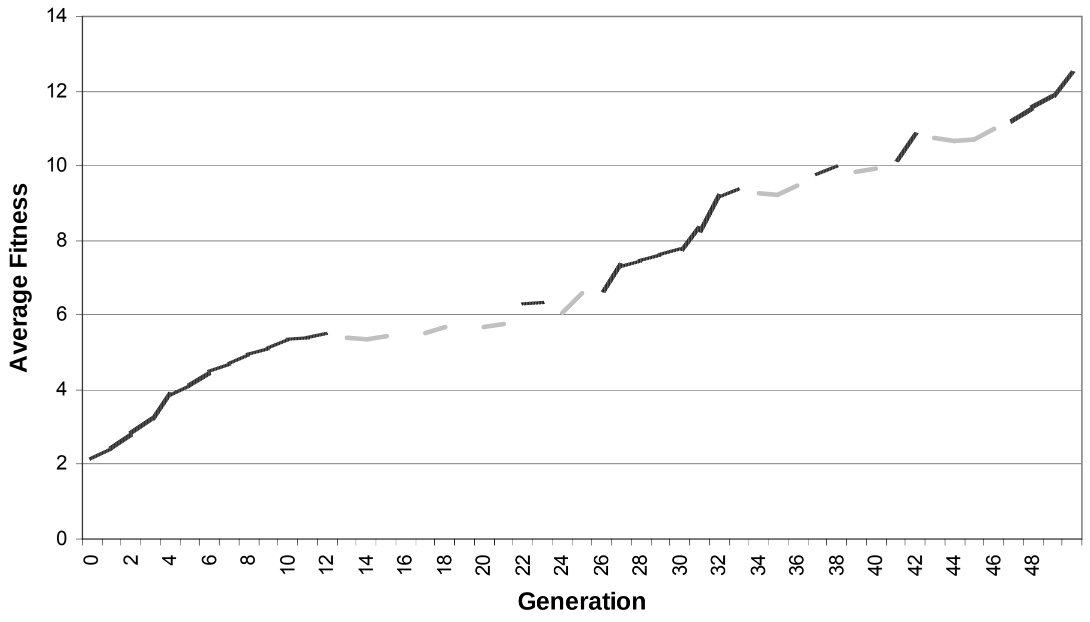


Figure 8.23 Average Fitness for DIFF using mean as characteristic measure - Deg-4 Regression Problem

In Figure 8.23, the black dotted line represents the periods where population reduction is occurring and solid bolded grey line represents stages within the cycle where population increment is taking place. Interestingly, the standard deviation follows a similar pattern, even though phenotypic diversity is not directly formulated into the characteristic measure. The reader is referred to Figure 8.24.

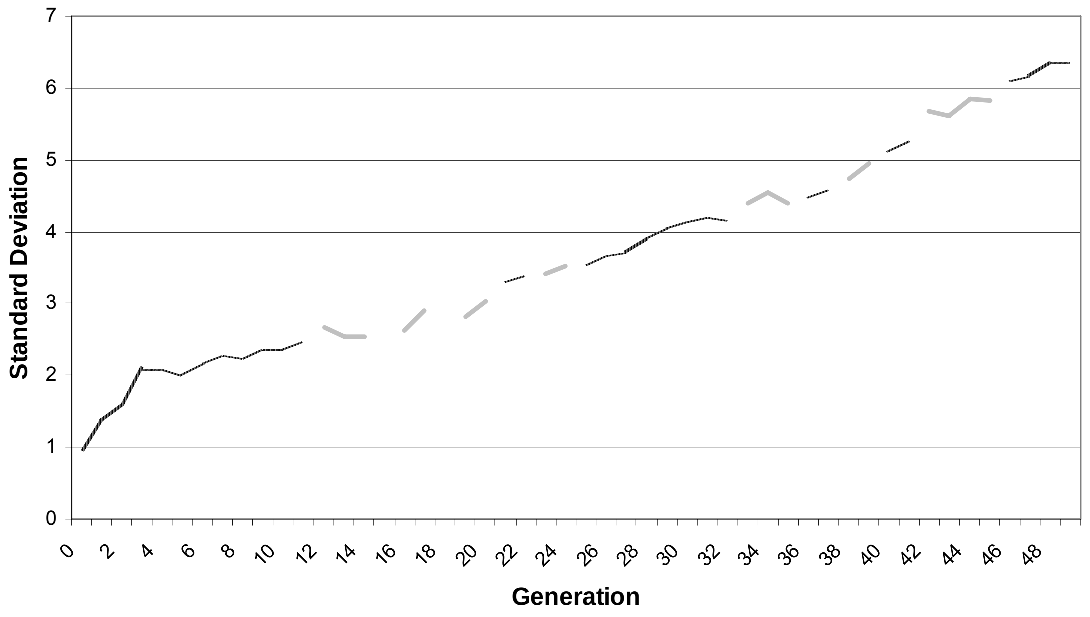


Figure 8.24 Fitness Standard Deviation for DIFF using mean as characteristic - Deg-4 Regression Problem

This highlights the verity that population variation has a close relationship with the phenotypic diversity. The entropy plot is also displayed in Figure 8.25. It should be noted that this graph does not follow the same pattern as in the last two Figures. The entropy function simply converges to a maximum possible level.

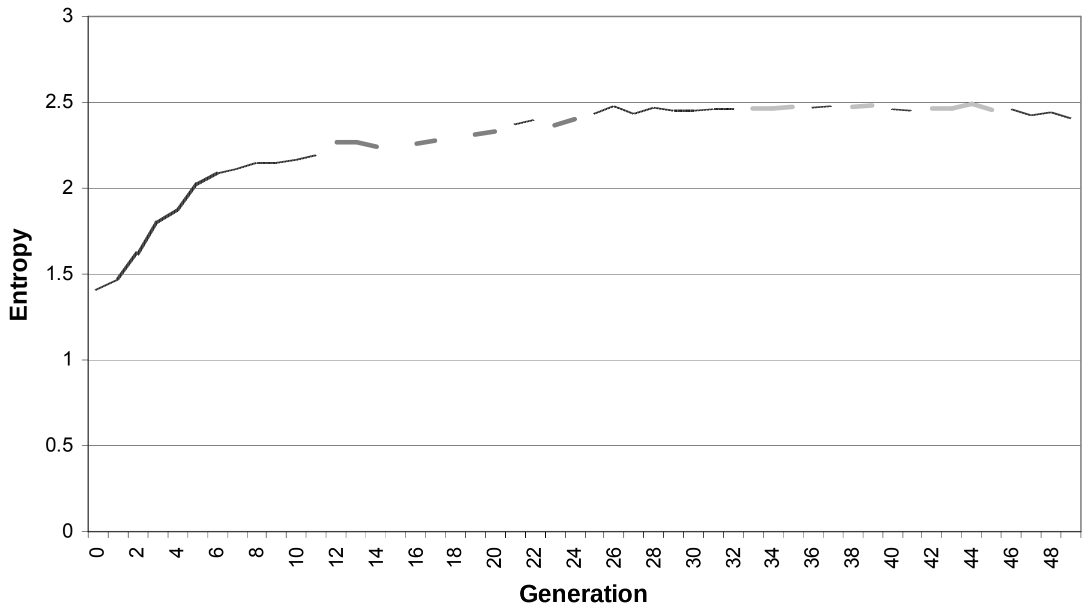


Figure 8.25 Entropy for DIFF using mean as characteristic measure - Deg-4 Regression Problem

The remarkable point is that the entropy function was converging towards a definite threshold. However, as further individuals were infused into the evolutionary environment, the horizontal asymptote was shifted further up as a result. This resulted in a 48% improvement in the threshold towards which the entropy function was converging.

The following note is needed to be accentuated. The best performing Dynamic Population Variation (DPV) regime can only be based on an optimal specific pivot function and a characteristic measure that best represents stagnation. The author believes a very robust pivot function could be formulated using the "Pivot Function Table of Actions" introduced in this chapter and by investigating which arrangements within this table will result in optimal performance, rather than investigating assorted forms of mathematical formulas for the pivot functions. Figure 8.26 summarises the cooperation with the evolutionary process required to obtain optimal results. It can be visualised that the identification of *stagnation* and *stagnation-beyond-repair* is of utmost importance. The term *stagnation-beyond-repair* refers to a stage within the evolutionary cycle, where no modification or action can revive the population set to bring it out of stagnation. At this stage, the best way forward would be abrupt termination. As the proposed pivot functions do not currently have this feature of abrupt termination, a ferocious reduction of individuals in the population set could create such a side-effect. In the following subsection, a brief investigation into the relationship of different measures and stagnation is undertaken.

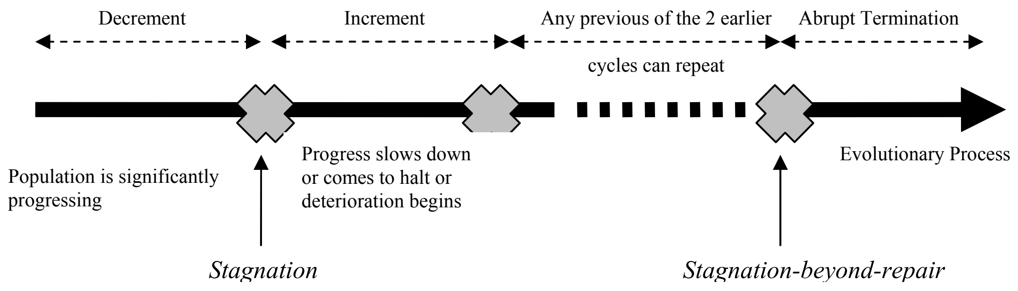

Decrement
Increment
Any previous of the 2 earlier
cycles can repeat
Abrupt Termination
Population is significantly progressing
Progress slows down or comes to halt or deterioration begins
Evolutionary Process
Stagnation
Stagnation-beyond-repair

Figure 8.26 The required cooperation with the evolutionary process necessary to obtain optimal results.

8.5.3 An Insight into Stagnation

In this subsection, various characteristic measures are examined to identify any possible trends that exist in their behaviour for runs where the run may be classified as an unsuccessful run perhaps due to depletion of rich individuals and hence resulting in stagnation. In addition, possible inclinations that may be present in successful runs for the same characteristic measures are looked at and comparisons are made between the two studies to recognise any emerging distinguishing patterns. One hundred independent runs are used in the study of this section. The successful runs are those, which approximately solve or 100% solve the problem, whereas runs which fail to reach the termination criterion are termed as unsuccessful runs. All the figures in this subsection are each comprised of three sets of graphs. The first graph shows the trends of the chosen characteristic measure in twenty selected successful runs, the middle graph presents the tendencies of the same measure in twenty unsuccessful runs and the final graph is the superimposing of the previous two graphs for ease of comparison. The first few characteristic measures looked at herein represent the best and worst individuals and their difference. In the former case, the measure is simply based on one individual within the population, but in the latter case on two individuals. The degree-4 Regression Problem is used in this study.

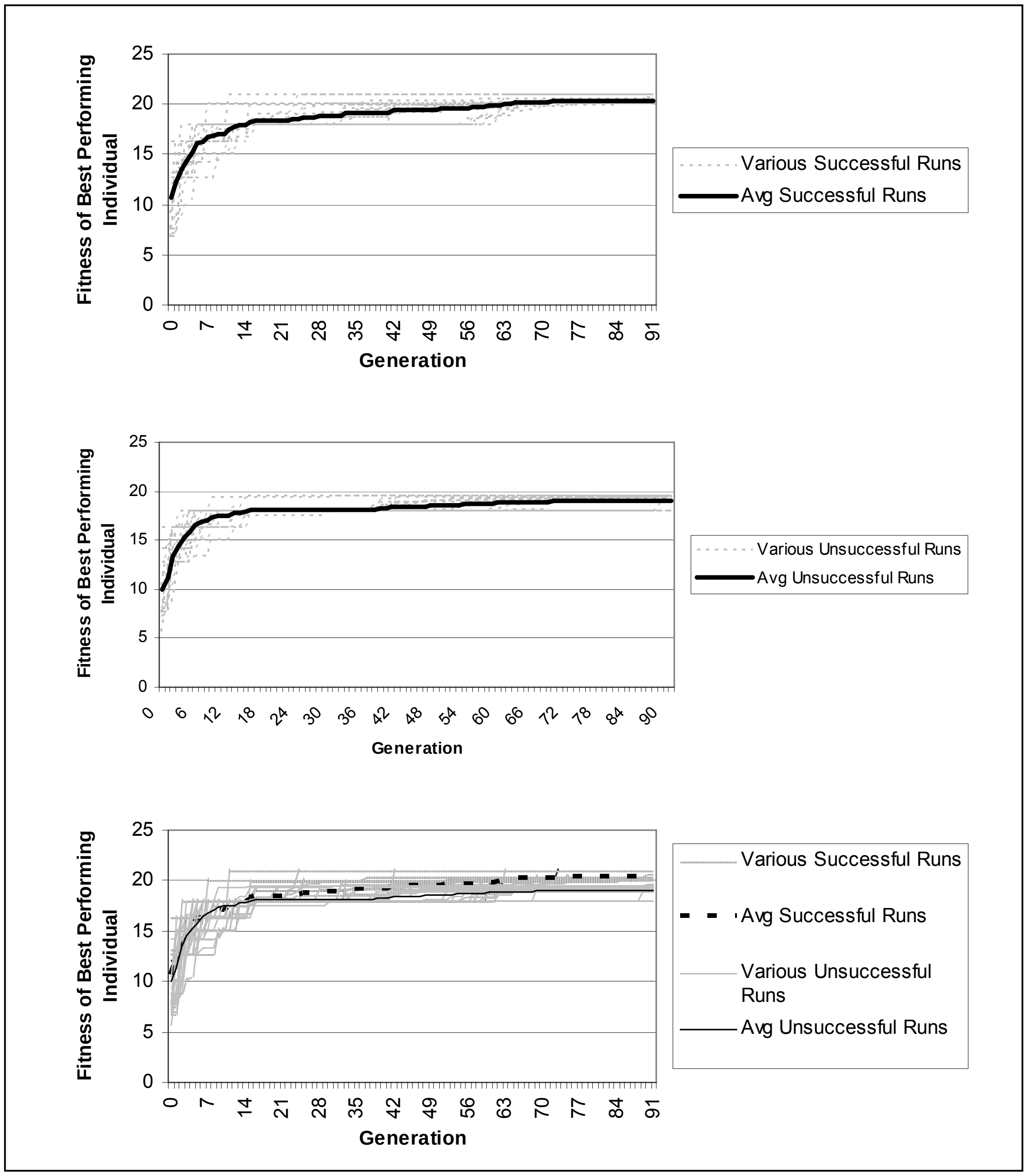


Figure 8.27 Fitness of best performing individual selected as characteristic measure

The first characteristic measure is the best performing individuals, which is shown in Figure 8.27. As it can be seen in Figure 8.27, both successful and unsuccessful runs show similar trends. The graphs are increasing functions ($\frac{df}{dg} \geq 0$), but the fitness grows slightly faster in successful runs than unsuccessful runs. Although the differences are quite small, the fitness of the best performing individual could be selected as a characteristic measure for measuring stagnation.

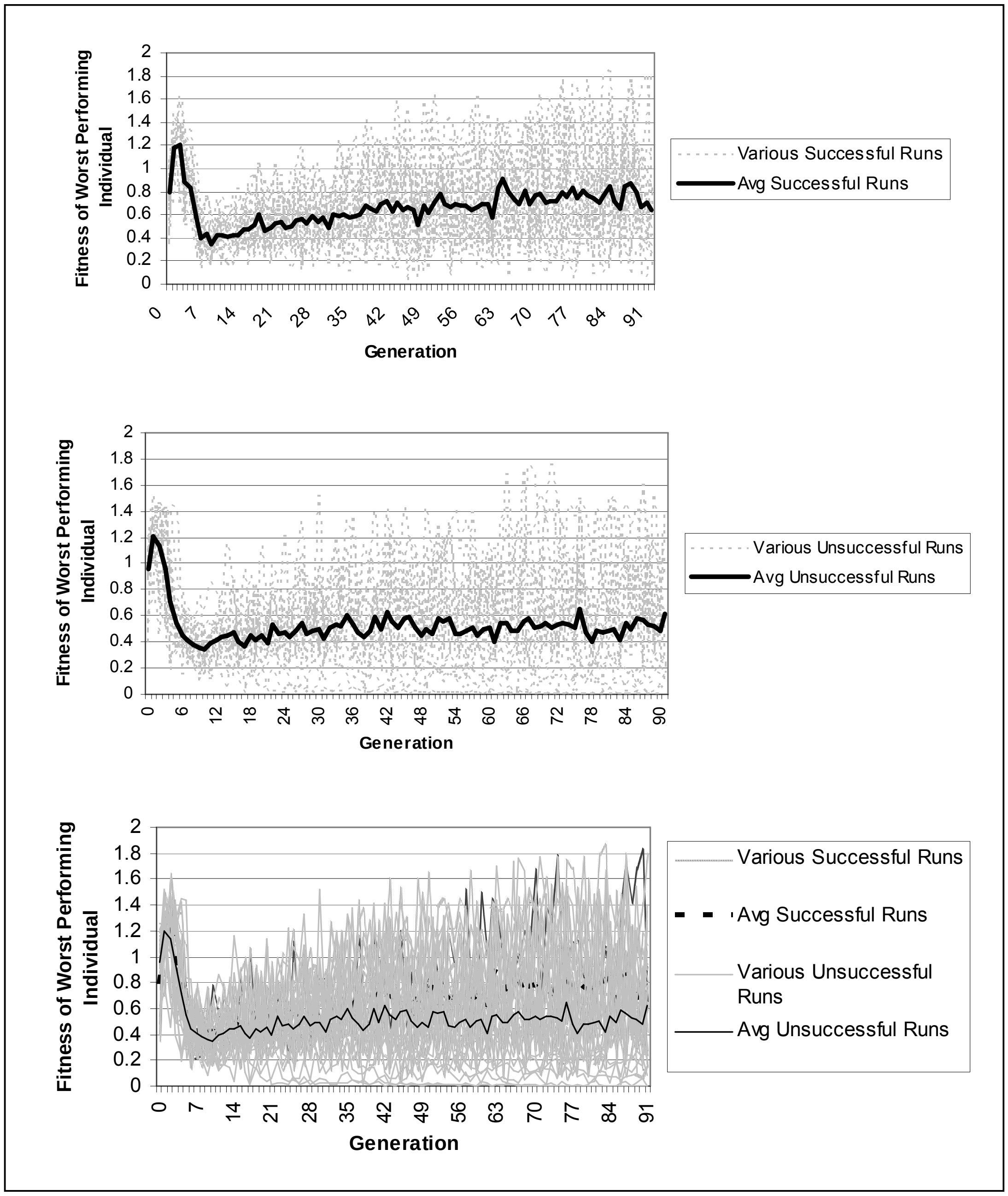


Figure 8.28 Fitness of worst performing individual selected as characteristic measure

No meaningful drift appears in the graph of the fitness of the worst performing individual (Figure 8.28). The fluctuations appear to be quite noisy as similar to Gaussian white noise, although its power spectral density here would not be completely flat. Therefore, the fitness of the worst performing individual should not be selected as a characteristic measure.

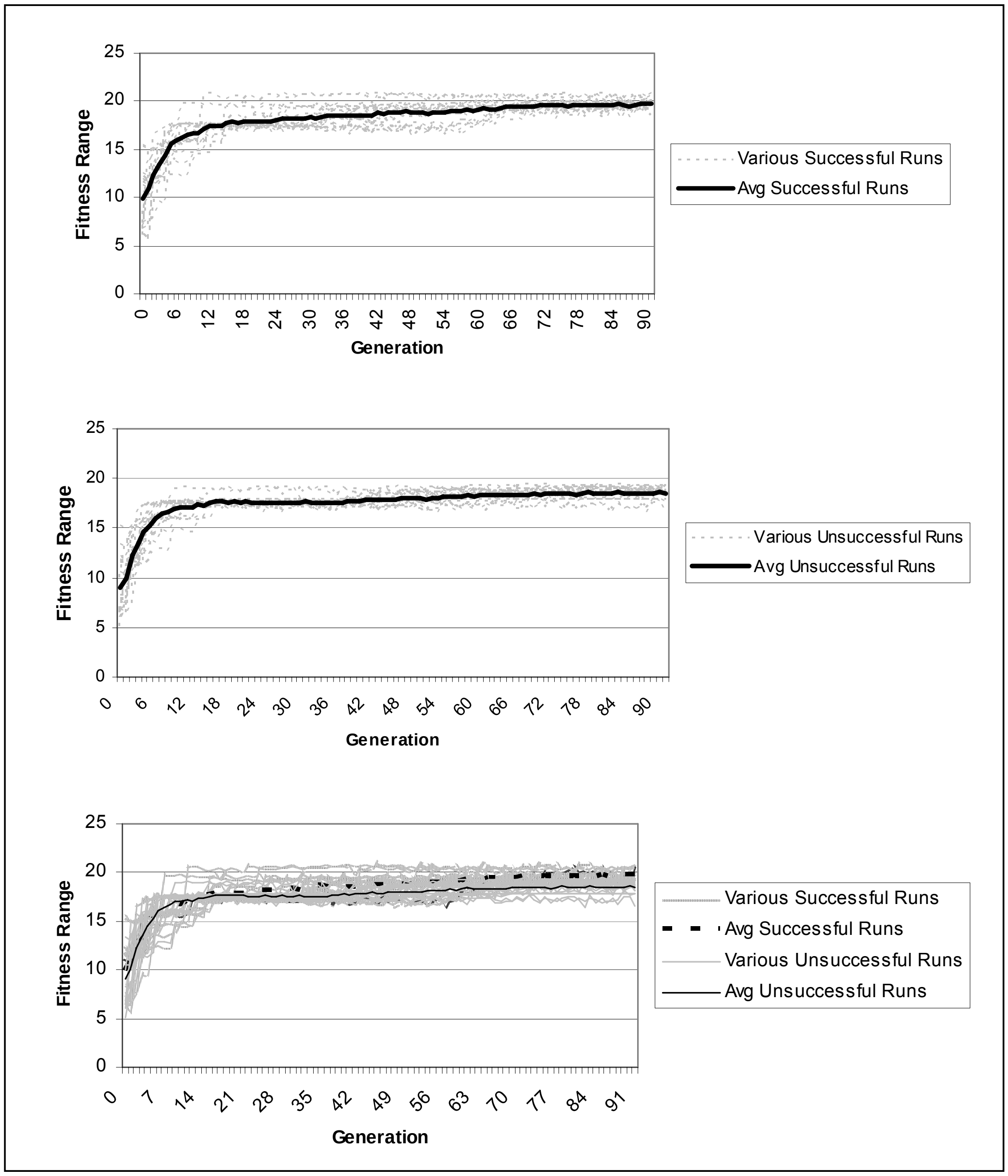


Figure 8.29 Fitness Range selected as characteristic measure

Although the fitness range (Figure 8.29) shows comparable inclinations to fitness of the best performing individual, it is preferable to select the fitness of the best performing individual as the characteristic measure over the fitness range, as it is not blurred by the noisy behaviour of the worst performing individuals. The next two characteristic measures are based also on one individual, but the individual concerned is not the extremes within the population set.

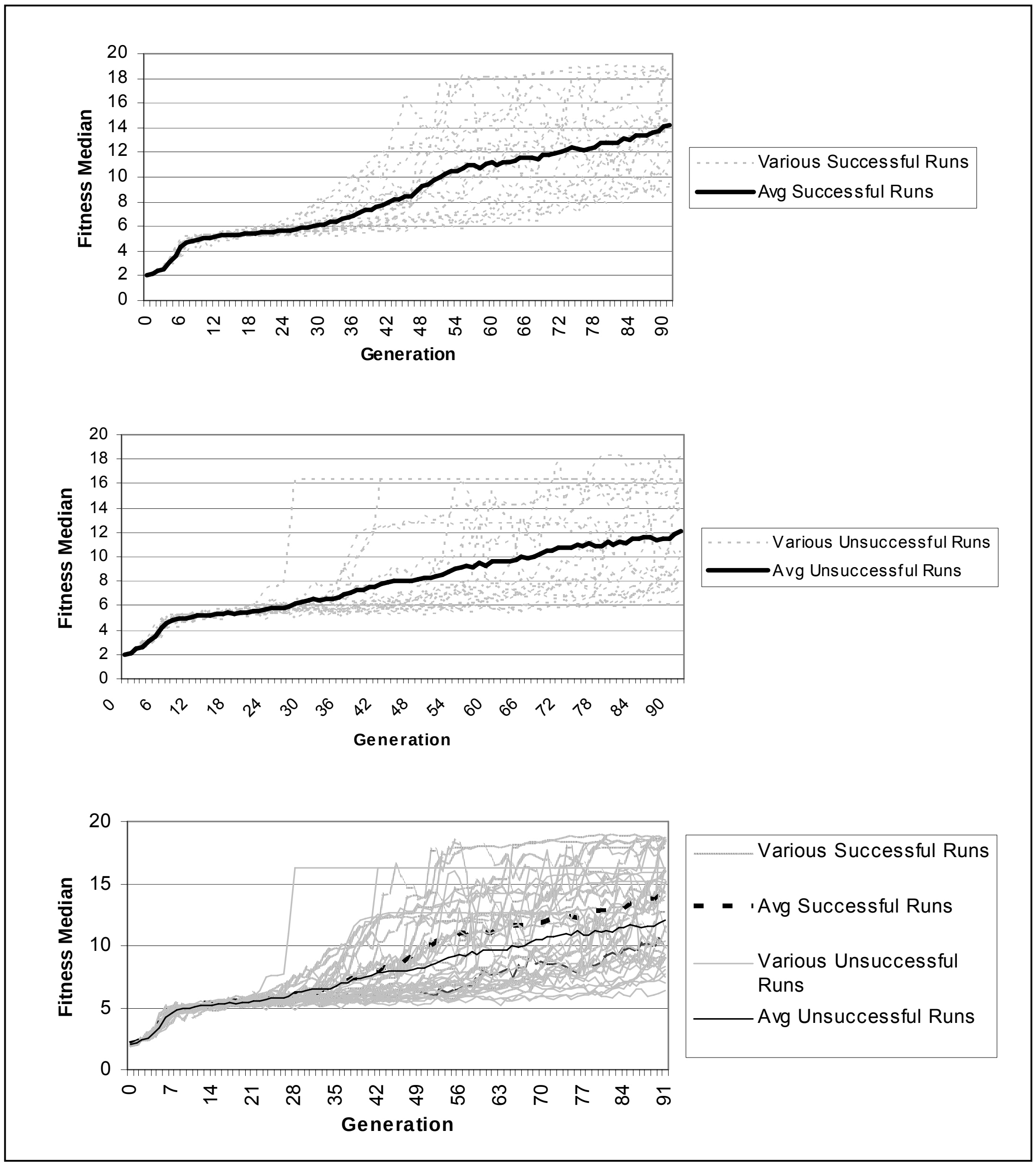


Figure 8.30 Fitness Median selected as characteristic measure

The median, which is a measure of central tendency, shows more significant differences (See Figure 8.30) between the successful and unsuccessful runs than the fitness of the best performing individuals. However, if each successful and unsuccessful graph is inspected separately, it can be seen that the changes are sometimes erratic. But the median on average appears to be the same in the beginning stages of the evolution and it is towards the end that it can be used as a distinguishing feature for separating successful runs from those which are not successful. Fitness UPQ (Figure 8.31) shows equivalent tendencies, but the changes are slightly less sporadic and in addition, the differentiation of successful runs from unsuccessful runs can commence earlier. It can be argued that UPQ may outperform the median measure.

The next two measures (fitness average and mean of high performing individuals) do not take into consideration merely a single individual for decision making. The latter uses a selected set of individuals, namely the cream of the crop or the elites, whereas the former utilises the entire population. Figure 8.32 shows the progress of average fitness as the evolutionary cycle is progressing. Average fitness shows promising trends for discriminating between successful and unsuccessful runs. However, it may be claimed that the mean fitness of the top performing individuals (Figure 8.33) could provide a better measure for stagnation as these individuals play a major role in progressing towards termination criterion than the set of the worst performing individuals. The progress of the lower performing individuals may not therefore be as critical and could distort the correct assessment of advancement of individuals or the stagnation of the evolutionary process.

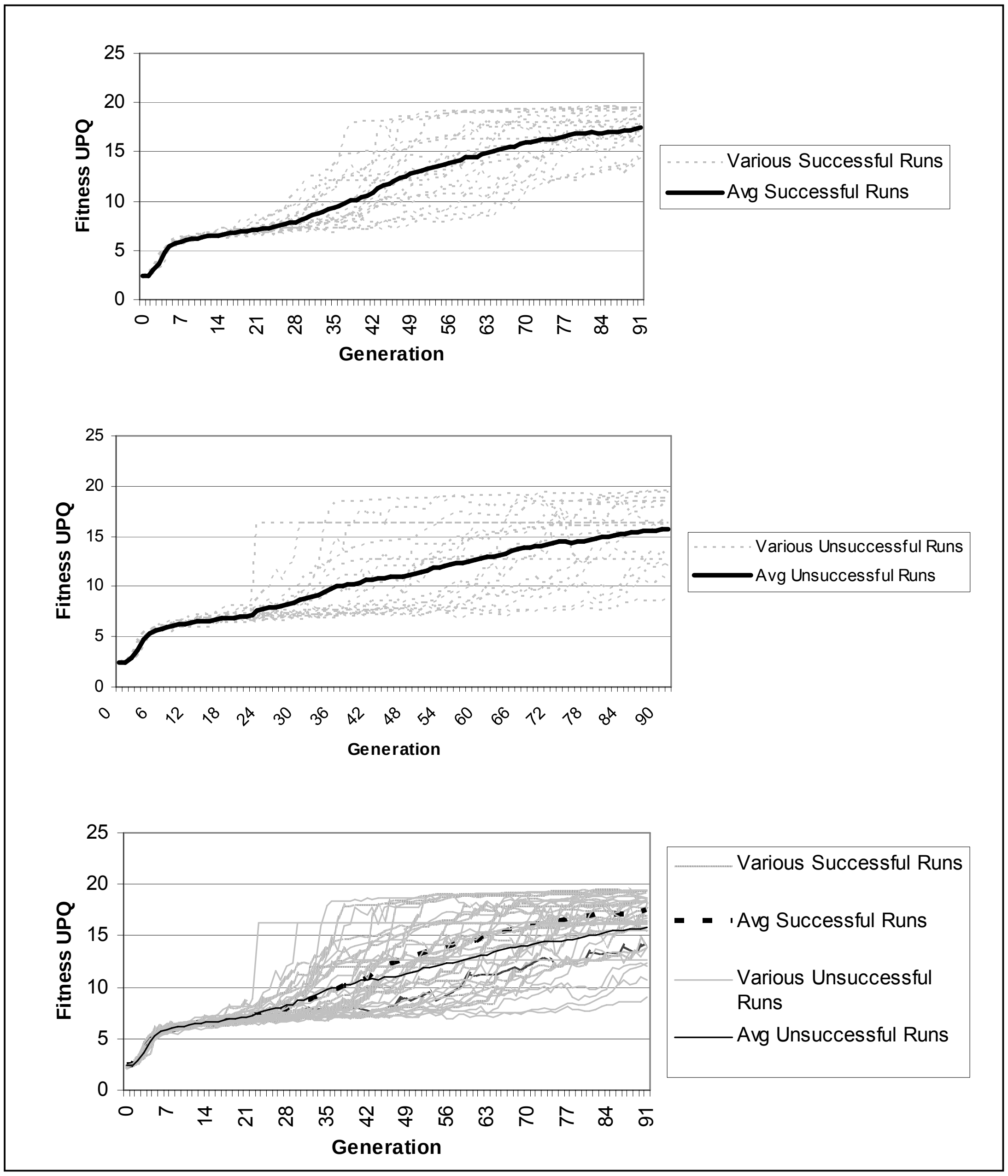


Figure 8.31 Fitness UPQ selected as characteristic measure

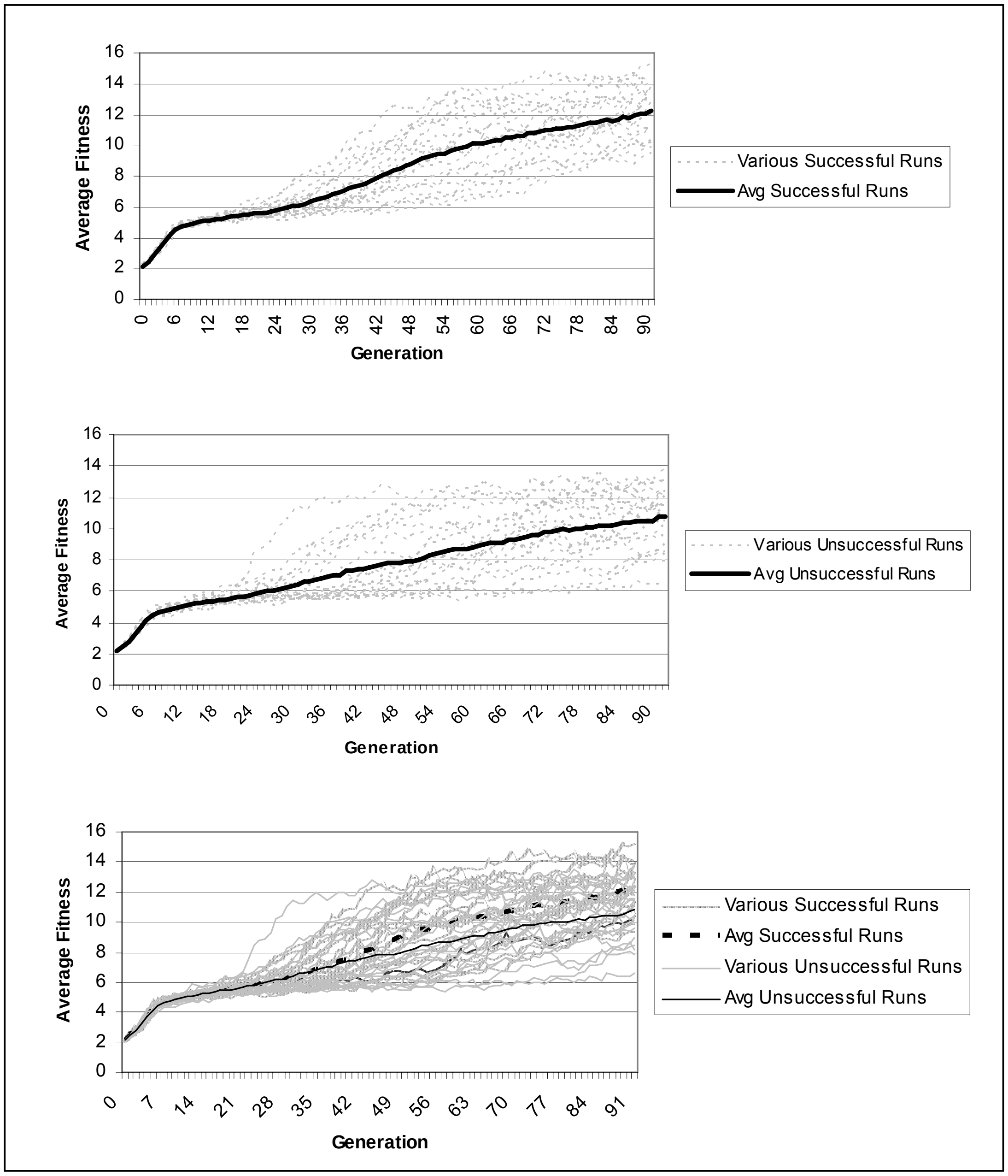


Figure 8.32 Average Fitness selected as characteristic measure

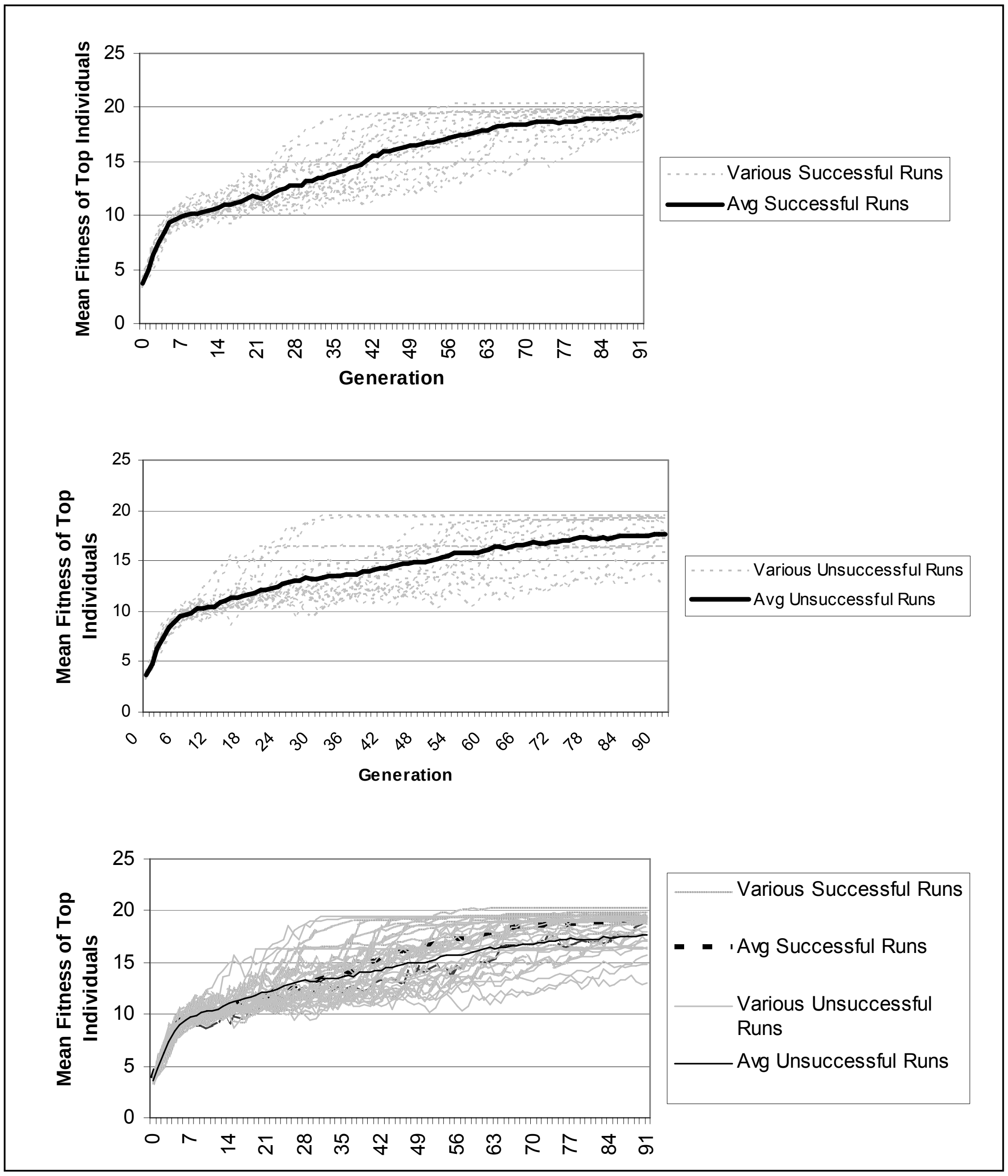


Figure 8.33 Mean Fitness of Top performing individuals selected as characteristic measure

The next two measures quantify the levels of phenotypic diversity. Figure 8.34 shows the standard deviation of fitness for the whole population. Figure 8.35 shows the entropy versus generation. Standard deviation on average shows an increasing behaviour. The standard deviation of the successful runs on average appears to be slightly larger and therefore this measure could be employed for determination of stagnation from phenotypic diversity point of view.

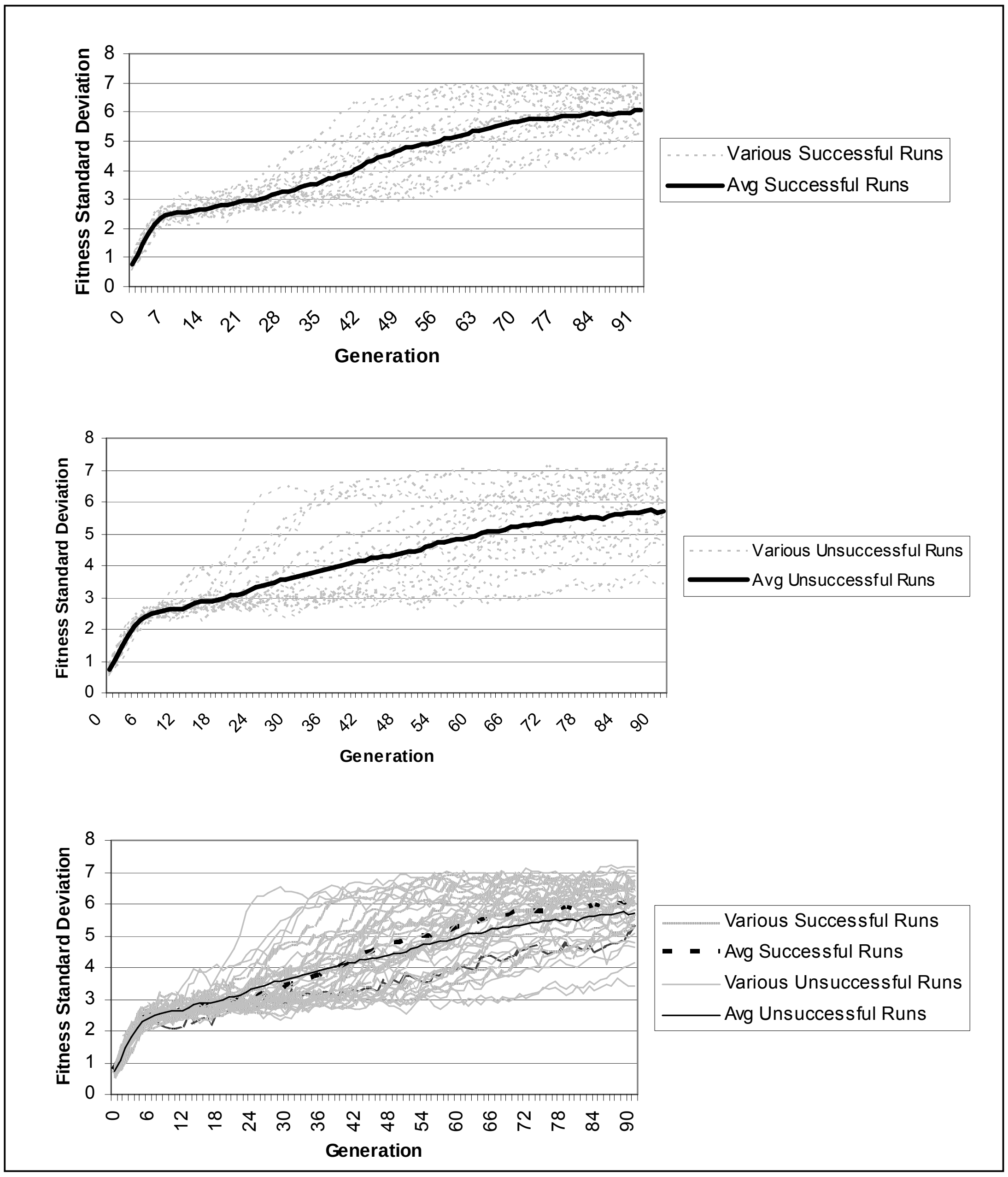


Figure 8.34 Fitness Standard Deviation selected as characteristic measure

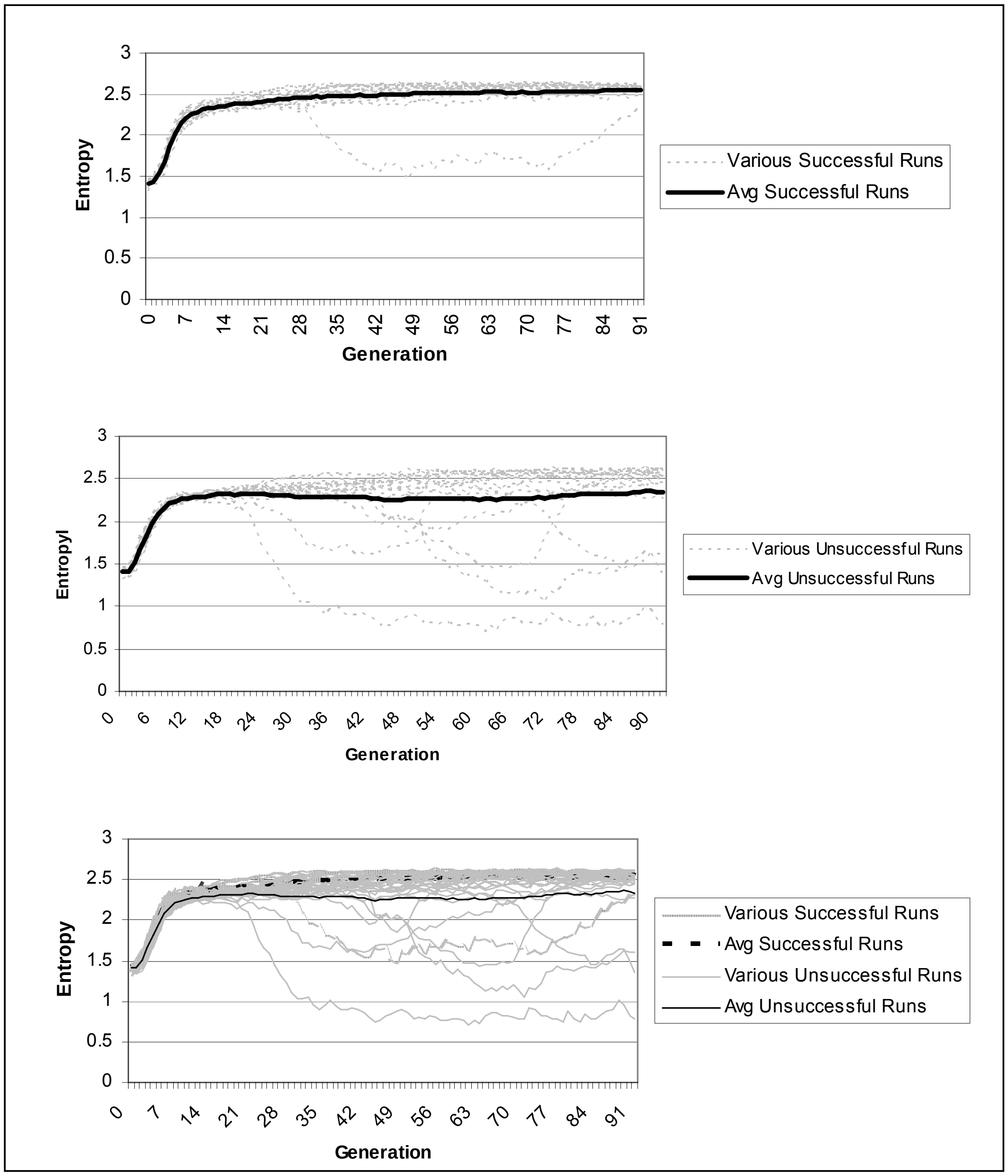


Figure 8.35 Entropy selected as characteristic measure

With only one exception, all successful runs exhibit an increasing function when entropy is the selected characteristics measure. However, the unsuccessful runs show a sudden decrease in levels of entropy at times. This behaviour can be exploited to identify stagnation. At these times, a proposed methodology for enriching the population set should be employed to bring the population out of stagnation.

In general, most of the measures studied in this subsection reveal their own merits for use as a characteristic measure for assessing the progress or stagnation of the evolutionary process. A whole study (in more detail and depth) devoted to this subject, taking into account the characteristic measures herein and newly defined measures, would strongly be recommended as a possible field of future research or study.

CHAPTER 9

CONCLUSION

“We don’t know one millionth of one percent about anything”, Thomas Edison.

9.1 Summary and Concluding Remarks

In the first chapter of this thesis, an overview of the Evolutionary Algorithms (EA) was presented. The chief components, which play a major role in the algorithm, were succinctly introduced. A brief literature survey of various research in the field of GP was presented. The fundamental principles of the Genetic Programming (GP) paradigm and its foundations are mainly based on the Genetic Algorithms (GA). For this reason, a brief peek into the Genetic Algorithms was inevitable. Thereafter, a thorough introduction into the theory of the Genetic Programming paradigm was launched. The majority of papers and books published have focused their efforts in examining or illustrating applications of the GP paradigm to various disciplines. Comparatively, very little research within the GP community has concentrated its efforts on devising schemes to improve the performance of GP. As a result, this work was primarily focused on investigating one possible method for improving the performance of GP. Subsequently, a comprehensive survey on all the current schemes proposed to improve the performance of GP was performed. These suggested modifications were then systematically categorised. This work took a glance at one way of improving the performance of Genetic Programming, namely population variation in GP.

Initially, the notion of static population variation was scrutinised. A new population variation (PV) scheme was recommended, whereby the size of the population was varied during the execution of the Genetic Programming system. Within this new scheme the initial population size was made to be different to the initial size of the standard Genetic Programming at its optimal or at least close to optimal starting population size. The population size was varied such that the worst case computational effort of the PV scheme was never greater than that of the standard Genetic Programming. Various schemes for altering population size under this proposal were propositioned to determine whether the nature of the "population variation", i.e. the way the population is varied during the search, has any significant impact on GP performance. Various reduction and increment profiles including a random profile were studied. Experimental evidence was produced indicating that population variation can significantly improve the performance of GP, provided that the fitness landscape was not plateaued. Moreover, it was demonstrated that every specific problem has its own most favourable profile.

Next, we delved into the concept of Dynamic Population Variation. Various innovations for dynamically varying the population size during the run of the Genetic Programming (GP) system were explored. The effectiveness of these innovations was investigated and it was confirmed that the new ideas do have the capacity to provide solutions at a lower computational effort compared with the standard genetic programming and previously reported algorithms, including the new PV algorithms.

In conclusion, this study produced experimental evidence that the static and dynamic population variations can both in general significantly improve the performance of the Genetic Programming system by reducing the computational effort to reach a given fitness level. The newly devised static Population Variation proved to be superior to the plague operator [79] and the newly proposed innovations for dynamic population variation demonstrated their superiority over standard Genetic Programming and all the population variation schemes previously reported [79] [81] [164].

9.2 Future Research

In this section, interesting research potentials for population variation are identified. Then, some of the open areas of research within the Genetic Programming paradigm together with possible and recommended future trends in this discipline are emphasised.

9.2.1 Future Research in Population Variation

An understanding and awareness of the timing or stage during the evolutionary process at which the population size is to be changed would form the ultimate prerequisite in devising the most powerful and exclusive population variation methodology. Although such a task may be signified as almost unattainable, a joint theoretical and/or empirical study in this topic may deem to be beneficial. This can further refine the relationship between stagnation and population variation. Further studies on population variation based on phenotypic diversity such as entropy or variance/standard deviation is also suggested. But then the question of “Does diversity play a more significant role at different stages of the evolutionary process?” may arise and need to be answered. It was concluded in [34] that variation in phenotypic diversity is problem dependant and its adjustment would likely be crucial at different stages of evolution. More importantly, it was concluded that the type and amount of diversity required at these different stages of evolution remains unclear. So a clearer understanding of the requirements for variation in diversity may need to be addressed first prior to engaging in such research. Next, further research on population variation based on a combined approach of using both the phenotypic diversity and the characteristic measures used in this work may be fruitful.

Supplementary investigations in different pivot functions are also advised. It is urged to explore all the various combinations of the “Pivot Table of Actions” for altering population size. It is believed that the ultimate and most powerful pivot function could be devised using the “Pivot Table of Actions” introduced in this work, rather than using some new forms of mathematical representations for the pivot functions.

In addition, the proposed PV schemes can be applied to the other population based algorithms, such as Evolutionary Strategies (ES) and Genetic Algorithms (GA).

9.2.2 Future Research in Genetic Programming

Some promising areas of future research, according to the author’s opinions, can be outlined as follows. A complete theoretical analysis of evaluating and measuring difficulty in GP could be pivotal. Further in-depth research in understanding the influence of diversity in various stages of evolutionary process in GP is called for. GP contains a phase of exploration followed by exploitation and the determination of type and the amount of diversity needed at these different evolutionary times is imperative. It is believed that exhaustive investigations into the issue of diversity will be worth pursuing in conjunction with performing some new systematic operation to remedy the lack of diversity.

Examination of different measures for stagnation and premature convergence could be most promising together with newly invented methodologies to either bring the population out of stagnation or abruptly terminating a run and commencing a new run with the gained experience and knowledge of the previous run, i.e. incorporated memory.

An understanding of how to highlight and flag sub-trees that are the main reason for success of the individual is crucial and new operators could be used to expose these highly competitive genes by keeping an updated genes library and incorporating them into the new individuals.

Extremely little research has been conducted in the field of implicit adaptation or self-adaptive control, i.e., modifications and parameter control are encoded as a genome and are evolved implicitly with the individual. The author believes that further breakthrough success may be achievable by exploring self-adaptive control.

Appendices

Appendix A – List of Figures

Appendix B – List of Tables

Appendix C – List of Symbols

C_i^e	The effective complexity of program *i*
C_i^a	The absolute complexity of program *i*
D	Depth of a tree
D_{initial}	Maximum initial tree depth
$D_{\text{evolution}}$	Maximum depth of a tree during the evolution
$\boldsymbol{F}$	Function Set
f_{ed}	Frequency of editing
f_i	The fitness of the individual *i*
f_i^e	The effective fitness of the individual *i*
g	Generation number
G	Maximum number of generations
$\boldsymbol{G}$	Genotype space
H	Schema/Schemata
$H(P)$	Entropy of a population *P*
K	Alphabet cardinality
L	Length of character string
m	Number of meta-symbols
M	Population size
NI	Number of individuals
NS	Number of schemata
$\boldsymbol{P}$	Phenotype space
P_{c} or p_{c}	Probability of crossover
$p_{\text{diff}}(g)$	The probability that the offspring produced by 1-point crossover between programs h and $\hat{h}$ does not match H, given that h matches H and $\hat{h}$ does not match H.
p_i^d	The probability of destructive crossover
P_{en}	Probability of encapsulation
P_{ip}	Probability of selecting an internal (function) point
P_{m} or p_{m}	Probability of mutation
P_{p}	Probability of permutation
P_{r}	Probability of reproduction
R	Number of independent runs
R_{ε}	Computational effort ratio
R_{λ}	AES ratio
R_{SR}	Success rate ratio
s	Standard Deviation
$\boldsymbol{S}$	Solution space
$\boldsymbol{T}$	Terminal Set
$\overline{X}$	Sample mean
*****	"Don't care" Symbol in a schema
ε	Computational Effort per run
ε_{c}	Probability of disruption due to crossover
ε_{m}	Probability of disruption due to mutation

ε_{PV}	Computational effort of the PV scheme
ε_{SGP}	Computational effort of SGP
λ	Average number of Evaluations to a Solution
Φ	The space of the population
Π	The multiset (set of all programs)
Ω	The space of all possible sub-expression in extractable from Φ

Appendix D – Acronyms and Abbreviations

ADF	Automatic Defined Functions
AES	Average number of Evaluations to a Solution
AI	Artificial Intelligence
AIM-GP	Automatic Induction of Machine Code with Genetic Programming
ALU	Arithmetic Logic Unit
APP	application nodes
AppGP	APP GP
AR-GP	Adaptive Representation GP
BDD	Binary Decision Diagrams
BMOPP	Biased Multi-Objective Parsimony Pressure
BNF	Backus Naur Form
CGP	Cartesian Genetic Programming
CFG	Context-Free Grammar
cGA	Compressed GA
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DAGS	Directed Acyclic Graphs
DGA	Double-based Genetic Algorithm
DNA	Deoxyribonucleic Acid
DNF	Disjunctive Normal Form
DPV	Dynamic Population Variation
DSP	Digital Signal Processing
EA	Evolutionary Algorithms
EAS	Elastic Artificial Selection
EC	Evolutionary Computation
EDI	Explicitly Defined Introns
EI	Evolutive Intron
EP	Evolutionary Programming
ES	Evolution Strategies
FCTS	Fully Covered Tournament Selection
FIFO	First-In-First-Out
FPS	Fitsness Proportionate Selection
FSM	Finite State Machines
GA	Genetic Algorithms
GA-P	Genetic Algorithms-Programming Hybrid
GBC	Grammar-based Crossover
GBIM	Grammar-based Initialisation Method
GBM	Grammar-based Mutation
GBX	Grammar-based Crossover
GDC	General Diversity Control
GE	Grammatical Evolution
GEGEP	Gene Estimated Gene Expression Programming
GEP	Gene Expression Programming
GGP	Graph genetic programming
GGGP	Grammar-guided genetic programming
GLiB	Genetic Library Builder
GMPE	Grammar Model-based Program Evolution

GNP	Genetic Network Programming
GP	Genetic Programming
GPNN	Genetic Programming Neural Network
GPP	Genetic Parallel Programming
HP	High Performing
IGP	Improved GP
ILP	Inductive Logic Programming
IMDL	Improved Minimum Description Length
IMGP	Isolated Multipopulation Genetic Programming
JFET	Junction Field Effect Transistor
LAGEP	Layered Genetic Programming
L-GP	Linearly structured GP
LISP	LISt Programming
LNO	Larger neutral offspring
MA	Memetic Algorithms
MA	Module Acquisition
MAP	Multi-ALU Processor
MDL	Minimum Description Length
MGP	Multi-population Genetic Programming
MHC	Maximum Homologous Crossover
ML	Machine Learning
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
MSE	Mean Square Error
NFL	No Free Lunch Theorem
NLNO	Non-larger neutral offspring
NN	Neural Networks
NNO	Non-neutral offspring
NOP	No Operations
PC	Personal Computer
PGP	Page-based genetic programming
PTC	Probabilistic Tree-Creation
PV	Population Variation
PV-ER	Population Variation-Exponential Reduction
PV-I	Population Variation-Increment
PV-LI	Population Variation-Linear Increment
PV-LR	Population Variation-Linear Reduction
PV-R	Population Variation-Reduction
PV-RAN	Random Population Variation
PV-SR	Population Variation-Step Reduction
PV-QR	Population Variation-Quadratic Reduction
RDC	Refined Diversity Control
SAMC	Self-Adaptive Multi-Crossover
SCPC	Strong Context Preservative Crossover
SGP	Standard Genetic Programming
SI	Self Improvement
SIM	Speciating Island Model
SNO	Smaller neutral offspring
SPEA	Strength Pareto Evolutionary Algorithm
SR	Success Rate
SRF	Sequence Referencing Function

SSAC	Selective Self-Adaptive Crossover
STROGANOFF	STructured Representation On Genetic Algorithms for Non-linear Function Fitting
STS	Standard Tournament Selection
TAG	Tree-adjoining Grammar
TAG3P+	Tree-adjoining Grammar Guided Genetic Programming
TGP	Traceless Genetic Programming
TM	Turing Machine
XOHC	Crossover with Hill Climbing
XOSA	Crossover with Simulated Annealing
UPQ	Upper quartile

Appendix E – Papers published as a result of this thesis

[i] Kouchakpour P, Zaknich A, Bräunl T, Population Variation in Genetic Programming, Information Sciences, Volume 177 , Issue 17, pp 3438-3452, 2007.

[ii] Kouchakpour P, Zaknich A, Bräunl T, Dynamic Population Variation in Genetic Programming, Information Sciences, submitted-in review.

[iii] Kouchakpour P, Zaknich A, Bräunl T, A Survey and Taxonomy of Performance Improvement of Canonical Genetic Programming, Knowledge and Information Systems, accepted, to be published.

Other papers which had previously been published by the author:

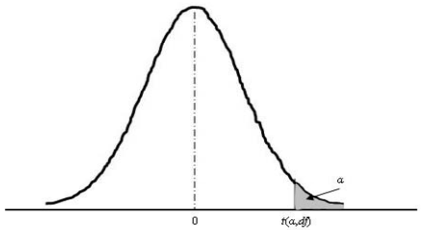
[iv] Kouchakpour P, Zomaya AY, Symbolic Computation of Robot Dynamic Parameters, Cybernetics and Systems 25 (5), pp 697-727, 1994.

Appendix F – List of Special Functions

a(f)	Arity of a function
$C(s \leftarrow k)$	The probability that crossover picks expression *s* in program *k*
$C(h,\hat{h})$	The set of indices of the crossover points in the common region between the program h and the program $\hat{h}$
$E[\,]$	The expected value operator
$f(H,g)$	Average fitness of a schema *H* at generation *g*
$\bar{f}(g)$	The average fitness of the population at generation *g*
$i(H,g)$	The number of instances of *H* at generation *g*
$l(H,k)$	Represents the schema obtained by replacing all of the nodes above point *k* with *
$L(H,k)$	The hyperschema produced by replacing the nodes between the crossover point k and the root node with * and all the sub-trees connected to those nodes with #
$m(H,g+1)$	Expected number of occurrences of schema *H* in the next generation
$N(j)$	The size of the program *j* matching the schema *H*
$N(H)$	The length of the schema *H* (the total number of nodes)
$NC(h,\hat{h})$	The number of nodes representing the common region between the program h and the program $\hat{h}$
O(H)	Schema Specificity
$P(M,i)$	Cumulative probability of satisfying the success predicate for generations between 0 and *i*.
PLOG	Protected logarithm
PROGN	Two arguments Connective function. The connective function executes the first argument and then the second argument in sequence.
PROGN3	3-arguments Connective function
PROGN4	4-arguments Connective function
$P(i \leftarrow j,s)$	The probability of inserting expression *s* in program *j* to create program *i*
$p(x, g)$	The probability of selection of program *x* at generation *g*
$p(H,g)$	The probability of selection of schema *H* at generation *g*
$p^d(H,j,g)$	The probability of disruption of schema *H* contained in program *j*
$R(z)$	Number of independent runs with a success probability of *z*
$SRF(n,d)$	Sequence Referencing Function - returns the value of the previously computed n^{th} term, provided $0 < n \le J-1$ (*J* is the index position) and otherwise it returns the default of *d*.
$u(H,k)$	Represents the schema obtained by replacing all of the nodes below point *k* with *
$U(H,k)$	The hyperschema produced by replacing the subtree below the crossover point k with #
$\delta(H)$	Defining length
$\delta(x)$	Returns 1 if *x* is *TRUE*, otherwise 0
%	Protected division operator

Appendix G – Critical Points of the t-Distribution

Critical Values of *t [202]*



For a particular number of degrees of freedom, entry represents the critical value of t corresponding to a specified upper-tail area (α)

UPPER-TAIL AREAS

Degrees of Freedom	**0.25**	**0.10**	**0.05**	**0.025**	**0.01**	**0.005**
1	1.0000	3.0777	6.3138	12.7062	31.8207	63.6574
2	0.8165	1.8856	2.9200	4.3027	6.9646	9.9248
3	0.7649	1.6377	2.3534	3.1824	4.5407	5.8409
4	0.7407	1.5332	2.1318	2.7764	3.7469	4.6041
5	0.7267	1.4759	2.0150	2.5706	3.3649	4.0322
6	0.7176	1.4398	1.9432	2.4469	3.1427	3.7074
7	0.7111	1.4149	1.8946	2.3646	2.9980	3.4995
8	0.7064	1.3968	1.8595	2.3060	2.8965	3.3554
9	0.7027	1.3830	1.8331	2.2622	2.8214	3.2498
10	0.6998	1.3722	1.8125	2.2281	2.7638	3.1693
11	0.6974	1.3634	1.7959	2.2010	2.7181	3.1058
12	0.6955	1.3562	1.7823	2.1788	2.6810	3.0545
13	0.6938	1.3502	1.7709	2.1604	2.6503	3.0123
14	0.6924	1.3450	1.7613	2.1448	2.6245	2.9768
15	0.6912	1.3406	1.7531	2.1315	2.6025	2.9467
16	0.6901	1.3368	1.7459	2.1199	2.5835	2.9208
17	0.6892	1.3334	1.7396	2.1098	2.5669	2.8982
18	0.6884	1.3304	1.7341	2.1009	2.5524	2.8784
19	0.6876	1.3277	1.7291	2.0930	2.5395	2.8609
20	0.6870	1.3253	1.7247	2.0860	2.5280	2.8453
21	0.6864	1.3232	1.7207	2.0796	2.5177	2.8314
22	0.6858	1.3212	1.7171	2.0739	2.5083	2.8188
23	0.6853	1.3195	1.7139	2.0687	2.4999	2.8073
24	0.6848	1.3178	1.7109	2.0639	2.4922	2.7969
25	0.6844	1.3163	1.7081	2.0595	2.4851	2.7874
26	0.6840	1.3150	1.7056	2.0555	2.4786	2.7787
27	0.6837	1.3137	1.7033	2.0518	2.4727	2.7707
28	0.6834	1.3125	1.7011	2.0484	2.4671	2.7633
29	0.6830	1.3114	1.6991	2.0452	2.4620	2.7564
30	0.6828	1.3104	1.6973	2.0423	2.4573	2.7500
31	0.6825	1.3095	1.6955	2.0395	2.4528	2.7740
32	0.6822	1.3086	1.6939	2.0369	2.4487	2.7385
33	0.6820	1.3077	1.6924	2.0345	2.4448	2.7333
34	0.6818	1.3070	1.6909	2.0322	2.4411	2.7284
35	0.6816	1.3062	1.6896	2.0301	2.4377	2.7238
36	0.6814	1.3055	1.6883	2.0281	2.4345	2.7195
37	0.6812	1.3049	1.6871	2.0262	2.4314	2.7154
38	0.6810	1.3042	1.6860	2.0244	2.4286	2.7116
39	0.6808	1.3036	1.6849	2.0227	2.4258	2.7079
40	0.6807	1.3031	1.6839	2.0211	2.4233	2.7045

Appendix H – Summary of Parameters for each Problem

Parameter	Value
Terminal Set	Regression Problem: $\boldsymbol{T}=\{X\}$ Even Parity Problem: $\boldsymbol{T}=\{X,Y,Z,W\}$ 6-Symmetry Problem: $\boldsymbol{T}=\{X,Y,Z,W,K,P\}$ Recursive Induction: $\boldsymbol{T}=\{0,1,2,3,J\}$ Simple Induction: $\boldsymbol{T}=\{0,1,2,J\}$ Artificial Ant problem: $\boldsymbol{F}$={MOVE,TURN-RIGHT,TURN-LEFT }

Function Set	Regression Problem: $\boldsymbol{F}=\{+,\times, -,\%\}$
	Even Parity Problem: $\boldsymbol{F}$= {NAND,NOR}
	6-Symmetry Problem: $\boldsymbol{F}$= {AND,OR,XNOR}
	Recursive Induction: $\boldsymbol{F}$={+,×, -,SRF}
	Simple Induction: $\boldsymbol{F}=\{+,\times, -\}$
	Santa Fe Trail: $\boldsymbol{F}$={IF-FOOD-AHEAD,PROGN}
	Los Altos Trail: $\boldsymbol{F}$={IF-FOOD-AHEAD,PROGN,PROGN3,PROGN4}

Number Fitness Cases	Constant creation regression problem: 20 Cubic regression problem: 20 Even parity problem: 16 Digital symmetry problem: 64 Recursive Induction: 20 Simple Induction: 20 Santa Fe Trail: 89 Los Altos Hills Trail: 157
No. of generations G	90
Probability of cross-over	90
Probability of reproduction	10
Probability of choosing internal point	90
Sequence Induction And Even Parity Problem (values the same as recursive problem)	<u>Initial Population Size</u> Recursive Problem: $P_{\text{PV-R}}(0)=7000$ and $P_{\text{PV-I}}(0)=3040$ $M=P_{\text{SGP}}(g)=P_{\text{plague}}(0)=5000$, $\Delta P(\Delta g)=-100$ Plague removes 100 individuals per generation. Simple Induction Problem: $P_{\text{PV-R}}(0)=600$ and $P_{\text{PV-I}}(0)=200$ $M=P_{\text{SGP}}(\text{g})=P_{\text{plague}}(0)=300$, $\Delta P(\Delta g)=-10$ Plague removes 10 individuals per generation. <u>Some other Parameters</u> Generally, PV-SR Step Size = 100. evenly spaced. $m_{\text{PV-ER}}=-0.01$ (exponential rate of reduction) Generally, $P(g)_{\text{PV-RAN}}$ was limited to ±25% of M with step change of ±20. a and b were set such that $P(G)=0.1M$
Artificial Ant Problem, Regression Problem and Boolean symmetry problem	<u>Initial Population Size</u> $P_{\text{PV-R}}(0)=900$ and $P_{\text{PV-I}}(0)=10$ $M=P_{\text{SGP}}(g)=P_{\text{plague}}(0)=500$, $\Delta P(\Delta g)=-10$ <u>Some other Parameters</u> Generally, PV-SR Step Size = 100, evenly spaced.. $m_{\text{PV-ER}}=-0.01$ (exponential rate of reduction) Generally, $P(g)_{\text{PV-RAN}}$ was limited to ±25% of M with step change of ±20. Plague removed 10 individuals.
Termination Criteria (A expression that has a fitness of:)	Constant creation regression problem: 20 Cubic regression problem: 20 Even parity problem: 16 Digital symmetry problem: 64 Recursive Induction: 20 Simple Induction: 20 Santa Fe Trail: 89 Los Altos Hills Trail: > 130
Max initial depth	6
Max depth after x-over	All: 10, Except Artificial Ant problem: 14
Basic Selection	Fitness Proportionate
Spousal Selection	Fitness Proportionate
Creation of initial Population	Ramped half-and-half
Elitist Strategy	TRUE

References

[1] Altenberg L, The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, Advances in Genetic Programming, chapter 3, MIT Press, pp 47–74, 1994.

[2] Altenbreg L, Emergent phenomena in genetic programming, in Sebald and Fogel, editors, Evolutionary Programming, Proceedings of the third annual conference, pp 233-241, 1994.

[3] Altenberg L, Genome Growth and the evolution of the genotype-phenotype map, in Banzhaf and Eckman, editors, Evolution as a Computational Process, 1995.

[4] Ando S, Sakamoto E, Iba H, Modelling genetic network by hybrid GP, Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02, vol. 1, pp 291 – 296, 2002.

[5] Andre D., Learning and upgrading rules for an OCR system using genetic programming" IEEE World Congress on Computational Intelligence, vol 1, pp 462-467, 1994.

[6] Andre D. and Teller A., A study in program response and the negative effects of introns in genetic programming, in Genetic Programming 1996: Proc. First Ann. Conf. J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (eds.) MIT Press: Cambridge, MA, pp. 12–20, 1996.

[7] Andre D., Koza J.R., Parallel genetic programming: a scalable implementation using the transputer network architecture, in: P.J. Angeline, K.E. Kinnear, Jr. (Eds.), Advances in Genetic Programing, vol. 2, The MIT Press, Cambridge, MA, USA, 1996 (Chapter 16).

[8] Andre D., Koza J.R., A parallel implementation of genetic programming that achieves super linear performance, Information Sciences, vol. 106 (3-4), pp 201-218, 1998.

[9] Andre D., Bennett F. H., Koza J. R., Keane M. A., On the theory of designing circuits using genetic programming and a minimum of domain knowledge", IEEE Evolutionary Computation Proceedings, pp 130–135, 1998.

[10] Angeline P J, Pollack J B, The evolutionary induction of subroutines. In The Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, pp 236-241, 1992.

[11] Angeline P J, Pollack J B, Competitive Environments Evolve Better Solutions for Complex Tasks, Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA), pp 264-270, 1993.

[12] Angeline P J, Pollack J B, Coevolving high-level representations. In Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, pp 55-72, 1994.

[13] Angeline P. J., Genetic Programming: A Current Snapshot, Proceedings of the Third Annual Conference on Evolutionary Programming, World Scientific, pp 224-232, 1994.

[14] Angeline P, An investigation into the sensitivity of genetic Programming to the frequency of leaf selection during subtree crossover, Proceeding of the first annual conference, Genetic Programming, pp 21-29, 1996.

[15] Angeline P J, Two self-adaptive crossover operators for genetic programming, Advances in Genetic Programming: Volume 2, P J Angeline and K E Kinnear, Eds. MIT Press, Cambridge, MA, pp 89-109, 1996.

[16] Angeline P, Multiple interacting programs: A representation for evolving complex behaviors. Cybernetics and Systems, pp 779-806, 1998.

[17] Antolík J, Hsu WH, Evolutionary tree genetic programming, GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp 1789-1790, 2005.

[18] Araujo L, Multiobjective genetic programming for natural language parsing and tagging, Parallel Problem Solving From Nature - PPSN IX, Proceedings Lecture Notes in Computer Science, pp 433-442, 2006.

[19] Ashlock W, Ashlock D, Single parent genetic programming, The 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp 1172–1179, 2005.

[20] Badran K MS, Rockett PI, The roles of diversity preservation and mutation in preventing population collapse in multiobjective genetic programming, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1551-1557, 2007.

[21] Banzhaf W, Nordin P, Keller R E, and Francone F D, Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications. Morgan Kaufmann Publishers; Heidelburg : Dpunkt-verlag, 1998.

[22] Banzhaf W., Some Considerations on the Reason for Bloat, Department of Computer Science, Dortmund University, Revised November 21, 2001.

[23] Beasley D., Bull D. R. and Martin R. R, An overview of genetic algorithms part 1 Fundamentals, Technical report, University of Purdue, 1993.

[24] Beldek U., Leblebicloglu K., Strategy creation, decomposition and distribution in particle navigation, Information Sciences, vol. 177 (3), pp 755-770, 2007.

[25] Bengio S., Bengio Y., Cloutier J., Use of genetic programming for the search of a new learning rule for neural networks, IEEE World Congress on Computational Intelligence, vol 1, pp 324–327, 1994.
[26] Besetti S, Soule T, Function choice, resiliency and growth in genetic programming, GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp 1771-1772, 2005.
[27] Bleuler S., Brack M., Thiele L., Zitzler E., Multiobjective genetic programming: reducing bloat using SPEA2", Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, pp 536-543, 2001.
[28] BlickleT, Thiele L, Genetic Programming and Redundancy. In Hopf, J. (ed): Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94), Saarbruicken, pp 33-38, 1994.
[29] Blickle T., Evolving compact solutions in genetic programming: A Case Study, in Parallel Problem Solving From Nature IV, Proc. Int. Conf. on Evol. Comput., Berlin, Germany, H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel (eds.), LNCS, vol. 1141, Springer-Verlag: Berlin, pp. 564–573, 1996.
[30] Böhm W. and Geyer-Schulz A., Exact uniform initialization for genetic programming. In: R.K. Belew and M. Bose, Editors, Foundations of Genetic Algorithms IV, Morgan Kaufmann, University of San Diego, CA, USA (1996) 379-407, 1996.
[31] Boryczka M, Czech ZJ, Wieczorek W, Ant colony programming for approximation problems, Genetic and Evolutionary Computation GECCO 2003 , PT 1, Proceedings Lecture Notes in Computer Science, pp142-143, 2003.
[32] Burke E, Gustafson S, Kendall G, A survey and analysis of diversity measures in genetic programming, in Proc. Genetic and Evolutionary Computation Conf., W. B. Langdon et al., Eds., pp 716–723, 2002.
[33] Burke E, Gustafson S, Kendall G, Krasnogor N, Advanced population diversity measures in genetic programming," in Proc. 7th Int. Conf. Parallel Problem Solving From Nature, vol. 2439, LNCS, J. J. M. Guervós et al., Eds., pp 341–350, 2002.
[34] Burke E. K., Gustafson S., Kendall G., Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness, IEEE Transactions on Evolutionary Computation, vol. 8, Issue: 1, pp 47–62, 2004.
[35] Cagnoni S, Rivero D, Vanneschi L, A purely evolutionary memetic algorithm as a first step towards symbiotic coevolution, The 2005 IEEE Congress on Evolutionary Computation, Vol. 2, pp 1156-1163, 2005.
[36] Cai XY, Smith SL, Tyrrell AM, Positional independence and recombination in Cartesian Genetic Programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 351-360, 2006.
[37] Cao H Q, Kang L I, Guo T. Chen Y P, de Garis H, A two-level hybrid evolutionary algorithm for modelling one-dimensional dynamic systems by higher-order ODE models, IEEE Transactions on Systems, Man and Cybernetics, Part B, vol. 30, Issue 2, pp 351–357, 2000.
[38] Carbajal S. G., Martinez F. G., Evolutive Introns: A Non-Costly Method of Using Introns in GP, Genetic Programming and Evolvable Machines, vol. 2, pp 111-122, 2001.
[39] Carlos A, Coello Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge and Information Systems, 1(3), pp 129-156, 1999.
[40] Cheang SM, Leung KS, Lee KH, Genetic parallel programming: Design and implementation, Evolutionary Computation 14 (2), pp 129-156, 2006.
[41] Chellapilla K., Evolving computer programs without subtree crossover, IEEE Transactions on Evolutionary Computation 1 (1997) (3), pp. 209–216. 1997.
[42] Chongstitvatana P., Improving robustness of robot programs generated by genetic programming for dynamic environments, Circuits and Systems, pp 523–526, 1998.
[43] Christensen S, Oppacher F, Solving the artificial ant on the Santa Fe trail problem in 20,696 fitness evaluations, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1574-1579, 2007.
[44] Coley David A, an Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific, 1999.
[45] Collard P, Segapeli J L, Using a double-based genetic algorithm on a population of computer programs, Tools with Artificial Intelligence, Proceedings, 6th International Conference, pp 418-424, 1994.
[46] Cormen T. H., Leiserson C. E. and Rivest R. L., Introduction to Algorithms, the MIT press, Cambridge, Massachusetts, ch. 19, pp 381-399, 1996.
[47] Couchet J, Manrique D, Rios J, Rodriguez-Paton A, Crossover and mutation operators for grammar-guided genetic programming, Soft Computing 11 (10), pp 943-955, 2007.
[48] Crawford-Marks R, Spector L, Size control via size fair genetic operators in the pushGP genetic programming system, Proceedings of the genetic and evolutionary computation Conference, New York, pp 733–739, 2002.
[49] D'haeseleer P. & Bluming J., Effects of locality in individual and population evolution, in Advances in Genetic Programming, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press, ch. 8, pp 177–198, 1994.

[50] D'Haesler P, Context preserving crossover in genetic programming, IEEE Proceedings of the 1994 World Congress on computational intelligence, Orlando, pp 1:379–407, 1994.
[51] Daida J. M., Bersano-Begey T. F., Ross S. J. and Vesecky J. F., Evolving feature-extraction algorithms: adapting genetic programming for image analysis in geoscience and remote sensing, Geoscience and Remote Sensing Symposium, vol 4, pp 2077-2079, 1996.
[52] Daida J. M., Bertram R. R., Stanhope S. A., Koo J. C., Chaudhary S. A., Chaudhary O. A. and Polito J. A. L., What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem Genetic Programming and Evolvable Machines, vol 2, pp 165-191, 2001.
[53] Daida JM and Hilss AM, Identifying Structural Mechanisms in Standard GP, in GECCO, E. Cantú-Paz, et al., 2003, Springer-Verlag, pp 1639–1651, 2003.
[54] Daida JM, Characterizing the dynamics of symmetry breaking in genetic programming, GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp 799-806, 2006.
[55] Darwin C., The origin of Species. John Murray, 1859.
[56] DeGaris H., Genetic programming: building nanobrains with genetically programmed neural network modules", 990 IJCNN International Joint Conference, pp 511–516, 1990.
[57] deGaris H, Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules. In Proc. Seventh International Conf. on Machine Learning (ICML-90), Porter, B.W. et al (eds), pp 132-139, 1990.
[58] de Jong E. D., Watson R. A., & Pollack J. B., Reducing bloat and promoting diversity using multi-objective methods," in Proc. Genetic Evolutionary Computation Conf., L. Spector et al., Eds., San Francisco, CA, pp 11–18, 2001.
[59] de Jong Edwin and Pollack Jordan, Multi-objective Methods for tree size control, Genetic Programming and Evolvable Machines, pp. 211-133, 2003.
[60] de Vega F. F., Roa L. M., Tomassini M., Sanchez J.M., Medical knowledge representation by means of multi-population genetic programming: an application to burn diagnosing, Engineering in Medicine and Biology Society, vol 1, pp. 619–622, 2000.
[61] de Vega F. F., Roa L.M., Tomassini M., Sanchez J.M., Multipopulation genetic programming applied to burn diagnosing, Evolutionary Computation, vol.2, pp 1292-1296, 2000.
[62] de Vega FF, Cantu-Paz E, Lopez JI, Manzano T, Saving resources with plagues in genetic algorithms, Parallel Problem Solving From Nature - PPSN VIII Lecture Notes in Computer Science, pp 272-281, 2004.
[63] de Vega FF, Gil GG, Pulido JAG, Guisado JL, Control of bloat in genetic programming by means of the island model, Parallel Problem Solving From Nature - PPSN VIII, Lecture Notes in Computer Science, pp 263-271, 2004.
[64] Deb K, Agrawal S, Pratab A, and Meyarivan T, A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II", Kan-GAL report 200001, Indian Institute of Technology, Kanpur, India (2000).
[65] Dignum S, Poli R, Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1588-1595, 2007.
[66] Dill K. M., Herzog J. H., Perkowski M. A., Genetic programming and its applications to the synthesis of digital logic Communications", Computers and Signal Processing, vol.2, pp 823-826, 1997.
[67] Dracopoulos D C and Kent S, Speeding up genetic programming: A parallel BSP implementation, in Proceedings of the First Annual Conference on Genetic Programming 1996, J.R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, July 28–31, pp 125–136, 1996.
[68] Du X, Li YQ, Xie DT, Kang LS, A new algorithm of automatic programming: GEGEP, Simulated Evolution and Learning, Proceedings Lecture Notes in Computer Science, pp 292-301, 2006.
[69] Dworman G., Kimbrough S. O. and Laing J. D., On automated discovery of models using genetic programming in game-theoretic contexts, System Sciences, vol.3, pp 428- 438, 1995.
[70] Eggermont J, Kok JN, Kosters WA, Detecting and pruning Introns for faster decision tree evolution, Parallel Problem Solving From Nature - PPSN VIII Lecture Notes in Computer Science, pp 1071-1080, 2004.
[71] Eiben A. E., Smith J. E., Introduction to Evolutionary Computing, Springer, Natural Computing Series, 1st edition, pp 129-151, 2003.
[72] Eiben A. E., Marchiori E and Valko V A, Evolutionary algorithms with the on-the-fly population size adjustment, In Xin Yao et al., Eds., Parallel Problem Solving from Nature-PPSN VIII, vol. 3242 of LNCS, pp 41-50, UK, Springer-Verlag, 2004.
[73] Ekárt A. & Németh S., A metric for genetic programs and fitness sharing, in Proc. European Conf. Genetic Programming, vol. 1802, LNCS, R. Poli et al., Eds., Edinburgh, U.K., pp 259–270, 2000.

[74] Eskridge BE, Hougen DF, Memetic crossover for genetic programming: Evolution through imitation, Genetic and Evolutionary Computation GECCO 2004, pt 2, Proceedings Lecture Notes in Computer Science, pp 459-470, 2004.

[75] Eskridge B E, Hougen D F, Imitating success: a memetic crossover operator for genetic programming, Congress on Evolutionary Computation, CEC2004, vol.1, pp 809 – 815, 2004.

[76] Fernandez F, Tomassini M, Punch WF, Sanchez JM, Experimental study of multipopulation Parallel Genetic Programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 283-293, 2000.

[77] Fernandez F, Tomassini M, Sanchez J M, Experimental study of isolated multipopulation genetic programming, 26th Annual Conference of the IEEE Industrial Electronics Society, vol.4, IECON 2000, pp 2672 – 2677, 2000.

[78] Fernandez F, Galeano G, Gomez J A, and Sanchez J M, Efficient use of computational resources in genetic programming: controlling the bloat phenomenon by means of the island model, IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference, vol.3, pp 2520–2524, 2002.

[79] Fernandez F., Tomassini M., Vanneschni L, Saving computational effort in genetic programming by means of plagues. Proceedings of the 2003 Congress on Evolutionary Computation, IEEE Press, vol. 3, pp 2042-2049, 2003.

[80] Fernandez F., Tomassini M., Vanneschi L., An empirical study of multipopulation genetic programming, Genetic Program. Evolvable Mach. 4 (1), pp 21–51, 2003.

[81] Fernandez F., Tomassini M., Vanneschni L and Cuendet J., A new Technique for Dynamic Size Populations in Genetic Programming. Congress on Evolutionary Computation (CEC'2004), IEEE 2004.

[82] Fernandez F, Martin A, Saving effort in parallel GP by means of plagues, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 269-278, 2004.

[83] Fernandez T, Virtual ramping of genetic programming populations, Genetic and Evolutionary Computation GECCO 2004, PT 2, Proceedings Lecture Notes in Computer Science, pp 471-482, 2004.

[84] Ferrer G. J. and Martin W. N., Using genetic programming to evolve board evaluation functions, Evolutionary Computation, vol.2, pp 747–752, 1995.

[85] Fillon C., Bartoli A., A divide & conquer strategy for improving efficiency and probability of success in genetic programming , Genetic Programming, Proceedings Lecture Notes in Computer Science, vol. 1, pp 13-23, 2006.

[86] Fogel L. J., Owens A. J., Walsh M. J., Artificial Intelligence through a simulation of evolution. In: A. Callahan, M. Maxfield, L.J. Fogel, Eds., Biophysics and Cybernetics Systems. Spartan, Washington DC, pp 131-156, 1965.

[87] Fogel L. J., Owens A. J., Walsh M. J., Artificial Intelligence through Simulated Evolution. Wiley, Chichester, UK, 1966.

[88] Folino G, Pizzuti C, Spezzano G, Vanneschi L, Tomassini M, Diversity analysis in cellular and multipopulation genetic programming, The 2003 Congress on Evolutionary Computation, CEC '03, vol. 1, pp 305–311, 2003.

[89] Folino G, Spezzano G,P-CAGE: An environment for evolutionary computation in peer-to-peer systems, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 341-350, 2006.

[90] Foster J., Maintaining the diversity of genetic programs, in Proc. 5th European Conf. Genetic Programming , vol. 2278, LNCS, et al., Eds., Kinsale, Ireland, pp 162–171, 2002.

[91] Freitag M N, Hopper N J, AppGP: an alternative structural representation for GP, Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, vol. 2, pp 1377-1383, 1999.

[92] Fry R, Tyrrell A, Enhancing the performance of GP using an ancestry-based mate selection scheme, Genetic and Evolutionary Computation GECCO 2003, pt 2, Proceedings Lecture Notes in Computer Science, pp 1804-1805, 2003.

[93] Fuchs M, Large Populations are not always the best choice in Genetic Programming, Proceedings of the Genetic and Evolutionary Computation Conference GECCO, pp 1033-1038, 1999.

[94] Fukunaga A S, Kahng A B, improving the performance of evolutionary optimization by dynamically scaling the evaluation function, Evolutionary Computation, pp182-187, 1995.

[95] Gagne C, Schoenauer M, Parizeau M, Tomassini M, Genetic programming, validation sets, and parsimony pressure Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 109-120, 2006.

[96] Galeano G, Fernandez F, Tomassini M, Vanneschi L, Studying the influence of synchronous and asynchronous parallel GP on programs length evolution, Proceedings of the Congress on Evolutionary Computation, CEC '02, vol. 2, pp 1727- 1732, 2002.

[97] Garcia S, Levine J, Gonzalez F, Multi niche parallel GP with a junk-code migration model, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 327-334, 2003.

[98] Garcia-Arnau M, Manrique D, Rios J, Rodriguez-Paton A, Initialization method for grammar-guided genetic programming, Knowledge-based Systems 20 (2), pp127-133, Mar 2007.

[99] Goldstein F. C., & Levin H. S., Disorders of reasoning and problem-solving ability. In M. Meier, A. Benton, & L. Diller (Eds.), Neuropsychological rehabilitation. London: Taylor & Francis Group, 1987.

[100] Guo H., Nandi A.K., Breast cancer diagnosis using genetic programming generated feature, IEEE Workshop on Machine Learning for Signal Processing, pp 215- 220, 2005.

[101] Gustafson S, Burke EK, The Speciating Island Model: An alternative parallel evolutionary algorithm, Journal of Parallel and Distributed Computing 66 (8), pp 1025-1036, 2006.

[102] Handley S., Automated learning of a detector for the cores of α-helices in protein sequences via genetic programming, World Congress on Computational, vol 1, pp 474–479, 1994.

[103] Handley S, on the use of a directed acyclic graph to represent a population of computer programs, in Proceedings of the IEEE Conference on Evolutionary Computation, pp. 154-159, 1994.

[104] Hansen J. V., Genetic Programming Experiments with Standard and Homologous Crossover Methods, Brigham Young University, Marriott School of Management, Provo, UT 84604, USA; Revised September 20, 2002.

[105] Hao H.T., Hoai N.X., McKay R.B., Does this matter where to start in grammar guided genetic programming?, Proceedings of the second Pacific Asian Workshop in Genetic Programming, Cairns, Australia, 2004.

[106] Harper R, Blair A, A Structure Preserving Crossover in Grammatical Evolution, IEEE Congress on Evolutionary Computation, pp 2537-2544, 2005.

[107] Harper R, Blair A, A Self-Selecting Crossover Operator, IEEE Congress on Evolutionary Computation, CEC 2006, pp 1420-1427, 2006.

[108] Harper R, Blair A, Dynamically Defined Functions In Grammatical Evolution, IEEE Congress on evolutionary Computation, CEC 2006, pp 2638 - 2645, 2006.

[109] Harries K. and Smith P., Exploring alternative operators and search strategies in genetic programming, in Genetic Programming 1997: Proc. Second Ann. Conf., Stanford University, CA, USA, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (eds.), Morgan Kaufmann, pp. 147–155, 1997.

[110] Hedberg S., Robots playing soccer? RoboCup poses a new set of AI research challenges, IEEE Intelligent Systems, Vol 12, pp 5–9, 1997.

[111] Hernandez-Aguirre A., Coello C. A., Buckles B. P., A genetic programming approach to logic function synthesis by means of multiplexers, Proceedings of the First NASA/DoD Workshop on Evolvable Hardware, pp 46-53, 1999.

[112] Heywood, M.J.; Zincir-Heywood, A.N., Page-based linear genetic programming, IEEE International Conference on Systems, Man, and Cybernetics, pp 3823 – 3828, 2000.

[113] Heywood MI, Zincir-Heywood AN, Dynamic page based crossover in linear genetic programming , IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics 32 (3), pp 380-388, 2002.

[114] Hinchliffe M, Hiden H, McKay B, Willis M, Tham M, Barton G, Modelling chemical process systems using a multi-gene genetic programming algorithm, In Koza, editor, Late Breaking papers at the Genetic Programming 1996 Conference, pp 56-65, 1996.

[115] Hirasawa K., Okubo M., Katagiri H., Hu J., Murata J., Comparison between Genetic Network Programming (GNP) and Genetic Programming (GP), Evolutionary Computation, vol. 2 , pp 1276–1282, 2001.

[116] Hoai N X, McKay R I, A Framework for tree Adjunct Grammar Guided Genetic Programming, Proceedings of the Post-graduate ADFA Conference on Computer Science (PACCS 01), pp 93-99, 2001.

[117] Hoai NX, McKay RI, Abbass HA, Tree adjoining grammars, language bias, and genetic programming, Genetic Programming , Proceedings Lecture Notes in Computer Science, pp 335-344, 2003.

[118] Holland J. H., Genetic Algorithms and the optimal allocation of trials. SIAM J. of Computing 2, pp.88-105, 1973.

[119] Holland J. H., Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, 1992. 1st edition: 1975, The University of Michigan Press, Ann Arbor, 1975.

[120] Hondo, N., Iba, H.,Kakazu, Y., Sharing and refinement for reusable subroutines of genetic programming, Proceedings of IEEE International Conference on Evolutionary Computation, pp 565 – 570, 1996.

[121] Hong J.H., Cho S.B., The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming, Artificial Intelligence In Medicine, vol. 36 (1), pp 43-58, 2006.

[122] Hong Yi, Ren Q and Zeng J, Adaptive population size for univariate marginal distribution algorithm, In IEEE congress on Evolutionary Computation, pp 1396-1402, 2005.

[123] Howard L M, D'Angelo D J, The GA-P: a genetic algorithm and genetic programming hybrid vol. 10, Issue 3, pp 11–15, 1995.
[124] Howard D. and Roberts S. C., Object detection by multiple textural analysers Evolutionary Computation, vol 2, pp 850-854, 1999.
[125] Hsu W H and Gustafon S M, Genetic Programming and Multi-Agent Layered Learning by Reinforcements. In Proc. GECCO 2002, pp 764-771, 2002.
[126] Hsu W H, Harmon S J, Rodriguez E and Zhong C, Empirical Comparison of Incremental Reuse Strategies in Genetic Programming for Keep-Away Soccer. In GECCO 2004 late-breaking papers, 2004.
[127] Iba H. Sato T., de Garis H., System identification approach to genetic programming, Evolutionary Computation, vol.1, pp 401-406, 1994.
[128] Iba H, deGaris H, Sato T, Genetic Programming using a Minimum Description Length Principle, in Advances in Genetic Programming, (ed. Kenneth E. Kinnear, Jr.) MIT Press, 1994.
[129] Iba H, deGaris H, Sato T, Temporal Data Processing Using Genetic Programming, in Proc. of 6th International Conference on Genetic Algorithms, pp 279-286, 1995.
[130] Iba H, Sato T, deGaris H, Recombination guidance for numerical genetic programming, IEEE International Conference on Evolutionary Computation, vol. 1, pp 97-102, 1995.
[131] Iba H and de Garis H, Extending genetic programming with recombinative guidance, Advances in Genetic Programming 2, P. J., Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA, USA: MIT Press, ch. 4, pp. 69–88, 1996.
[132] Iba H, Random Tree Generation for Genetic Programming, Source Lecture Notes In Computer Science; Vol. 1141, Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, pp 144-153, 1996.
[133] Iba H., Sasaki T., Using genetic programming to predict financial data Evolutionary Computation, Vol. 1, pp 244-251, 1999.
[134] Igel C., Kreutz M, Using fitness distributions to improve the evolution of learning structures, Evolutionary Computation, vol. 3, pp 1902-1909, 1999.
[135] Imae J., Nakatani S., Takahashi J., A design method for optimal controllers of minimax problems: a genetic programming approach, American Control Conference, vol.6, pp.5394–5399, 2003.
[136] Imae J, Kikuchi Y, Ohtsuki N, Kobayashi T, Guisheng Zhai, Design of nonlinear control systems by means of differential genetic programming, 43rd IEEE Conference on Decision and Control, CDC, vol. 3,pp 2734-2739, 2004.
[137] Ishida C Y, Pozo A, Grammatically based genetic programming for mining relational databases, Proceedings. 23rd International Conference of the Chilean Computer Science Society, SCCC 2003, pp 86 – 95, 2003.
[138] Ito T, Iba H, Sato S, Depth-dependent crossover for genetic programming, Evolutionary Computation Proceedings, IEEE World Congress, The 1998 IEEE International Conference on Computational Intelligence, pp 775–780, 1998.
[139] Ito T, Iba H, and Sato S, Non-destructive depth-dependent crossover for genetic programming. In Proceedings of the First European Workshop on Genetic Programming, vol 1391 of LNCS, Springer-Verlag, pp 71–82, 1998.
[140] Jackson D, Fitness evaluation avoidance in Boolean GP problems, The 2005 IEEE Congress on Evolutionary Computation, vol. 3, pp 2530-2536, 2005.
[141] Jackson D and Gibbons A P, Layered Learning in Boolean GP Problems. In Proc. EuroGP 2007, Lecture Notes in Computer Science, vol. 4445, Springer-Verlag, pp 148-159, 2007.
[142] Jackson D, Hierarchical genetic programming based on test input subsets, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1612-1619, 2007.
[143] Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C.E., Wang A., The Genesys system: Evolution as a theme in artificial life. Proceedings of Second Conference on Artificial Life (1992), edited by C. G. Langton and D Farmer. Redwood City, Artificial Life II, pp 549-578. Addison-Wesley, 1992.
[144] Jessen Yu, Keane M. A, Koza J. R., Automatic design of both topology and tuning of a common parameterized controller for two families of plants using genetic programming, Computer-Aided Control System Design, pp 234–242, 2000.
[145] Joshi A K, Levy L S, and Takahashi M, Tree adjunct grammars, Journal of Computer and System Sciences, vol. 10, pp. 136–163, 1975.
[146] Juillé H, Pollack J B, Massively parallel genetic programming, in Advances in Genetic Programming 2, P. J. Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA: MIT Press, ch. 17, pp. 339–358, 1996.
[147] Kaboudan M. A., Compumetric forecasting of crude oil prices, Evolutionary Computation, vol. 1, pp.283–287, 2001.

[148] Kalganova T and Miller J F, Evolving more efficient digital circuits by allowing circuit layout evolution and multiobjective fitness. In AStoica, D. Keymeulen, and I. Lohn, editors, Proceedings of the 1st NASMDoD Workshop on Evolvable Hardware (EH'99), IEEE Computer Society Press, pp 54-63, 1999.

[149] Kantschik W, Banzhaf W, Linear-tree GP and its comparison with other GP structures, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 302-312, 2001.

[150] Katagami D., Yamada S., Speedup of evolutionary behavior learning with crossover depending on the usage frequency of a node, IEEE International Conference on Systems, Man, and Cybernetics, IEEE SMC '99 Conference Proceedings, vol.5, pp 601-606, 1999.

[151] Keijzer M. , Efficiently representing populations in genetic programming, in Advances in Genetic Programming 2, P. J. Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA: MIT Press, ch. 13, pp 259–278, 1996.

[152] Keijzer M, Ryan C, O'Neill M, Cattolico M, Babovic V, Ripple crossover in genetic programming, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 74-86, 2001.

[153] Keijzer M, Alternatives in subtree caching for genetic programming. In Genetic Programming 7th European Conference, EuroGP 2004, Proceedings, vol 3003 of LNCS, Springer-Verlag, pp 328–337, 2004.

[154] Keith M J., Martin M. C., Genetic Programming in C++; Implementation issues. Chapter 13 Advances in Genetic Programming by Kenneth E. Kinnear, pp 285-310, 1994.

[155] Kennedy CJ and Giraud-Carrier C. A depth controlling strategy for strongly typed evolutionary programming. In W. Banzhaf, J. Daida, A. E. Eiben, et al., editors, GECCO-1999,Morgan Kaufman, pp 1–6, 1999.

[156] Kessler M, Haynes T, Depth-fair crossover in genetic programming, SAC '99: Proceedings of the 1999 ACM symposium on Applied computing, pp 319-323, 1999.

[157] Kim J J, Zhang B T, Effects of selection schemes in genetic programming for time series prediction, Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, Volume 1, pp 252-258, 1999.

[158] Kim M, Hiroyasu T, Miki M, SPEA2+: Improving the Performance of the Strength Pareto Evolutionary Algorithm2, Parallel Problem Solving from Nature - PPSN VIII, pp 742-751, 2004.

[159] Kinnear K. E. Jr., Evolving a sort: lessons in genetic programming, IEEE International Conference, vol 2, pp 881 – 888, 1993.

[160] Kinnear K. E. Jr., Fitness landscapes and difficulty in genetic programming, Evolutionary Computation, vol.1, pp 142–147, 1994.

[161] Kinnear K. E., Alternatives in Automatic Function Definition, Comparison of Performance in Advances in Genetic Programming, MA: MIT Press, pp 119 – 141, 1994.

[162] Korenaga M., Hagiwara M., Modified genetic programming based on elastic artificial selection and improved minimum description length, Systems, Man, and Cybernetics, vol. 3, pp 2348–2353, 1998.

[163] Kotani M., Nakai M. and Akazawa K., Feature extraction using evolutionary Computation, Evolutionary Computation, vol 2, pp 1230-1236, 1999.

[164] Kouchakpour P, Zaknich A, Bräunl T, Population Variation in Genetic Programming, Information Sciences, Volume 177 , Issue 17, pp 3438-3452, 2007.

[165] Kouchakpour P, Zaknich A, Bräunl T, Dynamic Population Variation in Genetic Programming, Information Sciences, submitted-in review.

[166] Kouchakpour P, Zaknich A, Bräunl T, A Survey and Taxonomy of Performance Improvement of Canonical Genetic Programming, Knowledge and Information Systems, submitted-in review.

[167] Koza J. R., Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. The 11th International Conference on Genetic Algorithms, ICGA, pp 768-774, 1989.

[168] Koza J.R., Genetically breeding populations of computer programs to solve problems in artificial intelligence, Proceedings of the 2nd International IEEE Conference, pp 819 – 827, 1990.

[169] Koza J. R. and Rice J. P., Genetic generation of both the weights and architecture for a neural network”, IJCNN-91-Seattle International Joint Conference, vol 2, pp 397–404, 1991.

[170] Koza J.R., Genetic Programming, On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.

[171] Koza J. R, A genetic approach to the truck backer upper problem and the inter-twined spiral problem, IJCNN, International Joint Conference, vol.4, pp 310-318, 1992.

[172] Koza J.R., Simultaneous discovery of detectors and a way of using the detectors via genetic programming, IEEE International Conference, vol 3, pp 1794-1801, 1993.

[173] Koza J. R., Keane M. A., Rice J. P., Performance improvement of machine learning via automatic discovery of facilitating functions as applied to a problem of symbolic system identification, IEEE International Conference, vol.1, pp 191–198, 1993.

[174] Koza J.R., Genetic Programming II, Automatic Discovery of Reusable Programs, MIT Press, Cambridge, MA, 1994.

[175] Koza J. R., Recognizing patterns in protein sequences using iteration-performing calculations in genetic programming Evolutionary Computation" , IEEE World Congress on Computational Intelligence, vol.1, pp 244–249, 1994 .

[176] Koza J. R, Automated discovery of detectors and iteration-performing calculations to recognize patterns in protein sequences using genetic programming computer Vision and Pattern Recognition", Proceedings CVPR, 1994 IEEE Computer Society Conference, pp 684–689, 1994.

[177] Koza J.R., Two Ways of Discovering the Size and Shape of a Computer Program to Solve a Problem, in ICGA, 1995, Morgan Kaufmann, pp 287–294, 1995.

[178] Koza J. R., Bennett F. H., Andre D., Keane M. A, Four problems for which a computer program evolved by genetic programming is competitive with human performance", Evolutionary Computation, pp 1 – 10, 1996.

[179] Koza J. R., Bennett F. H, Andre D., Keane M. A, Dunlap F., Automated synthesis of analog electrical circuits by means of genetic programming", Evolutionary Computation, pp 109–128, 1997.

[180] Koza J. R, Bennett F. H., Lohn J., Dunlap F., Keane M. A., Andre D., Automated synthesis of computational circuits using genetic programming", Evolutionary Computation, pp 447–452, 1997.

[181] Koza J. R., Bennett F. H., Andre D., Classifying proteins as extra cellular using programmatic motifs and genetic programming, Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, pp 212–217, 1998.

[182] Koza J.R., Bennett F. H., Andre D., Genetic Programming III, Darwinian Invention and Problem Solving, San Francisco, CA:Morgan Kaufmann, 1999.

[183] Koza J. R., Keane M. A., Yu J., Bennett F. H., Mydlowec W., Stiffelman O., Automatic synthesis of both the topology and parameters for a robust controller for a non-minimal phase plant and a three-lag plant by means of genetic programming", Decision and Control, Vol 5, pp 5292–5300, 1999.

[184] Koza J.R., Keane M.A., Streeter M.J., et al., Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer Academic Publishers, Norwell, MA, 2005.

[185] Kraft D. H., Petry F. E., Buckles B. P., Sadasivan T., The use of genetic programming to build queries for information retrieval", IEEE World Congress on Computational Intelligence, vol.1, pp 468–473, 1994.

[186] Kramer M.D., Du Zhang, GAPS: a genetic programming system, The 24th Annual International Computer Software and Applications Conference, COMPSAC 2000, pp 614 - 619, 2000.

[187] Kurashige K., Fukuda T., Hoshino H., Motion planning based on hierarchical knowledge for six legged locomotion robot", Systems, Man, and Cybernetics, vol 6, pp 924–929, 1999.

[188] Lang K J, Hill climbing beats genetic search on a Boolean circuit synthesis of Koza's, in Proceedings of the 12th International Conference on Machine Learning, pp. 340-343, 1995.

[189] Langdon W. B. , Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!. Norwell, MA: Kluwer, vol. 1, Genetic Programming, 1998.

[190] Langdon W. B., The evolution of size in variable length representations, Evolutionary Computation, pp 633–638, 1998.

[191] Langdon WB, Poli R, Fitness causes bloat: Mutation. In 1st European Workshop on Genetic. Programming, Springer-Verlag, pp 37–48, 1998.

[192] Langdon WB, Size fair and homologous tree genetic programming crossovers. In: Proceedings genetic and evolutionary computation conference, GECCO-99, Washington DC, pp 1092–1097, 1999.

[193] Langdon WB, Soule T, Poli R, and Foster JA. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline, editors, Advances in Genetic Programming III, MIT Press, pp 163–190, 1999.

[194] Langdon WB, Size fair and homologous tree genetic programming crossovers. Genetic Programming And Evolvable Machines, 1(1/2), pp 95–119, 2000.

[195] Langdon WB, Poli R, Foundations of Genetic Programming, Springer-Verlag, Berlin Heidelberg, 2002.

[196] Lasarczyk CWG, Dittrich P, Banzhaf W, Dynamic subset selection based on a fitness case topology Evolutionary Computation 12 (2), pp 223-242, 2004.

[197] Lee CY, Yao X, Evolutionary programming using mutations based on the Lévy probability distribution. IEEE Transaction Evolutionary Computation 8(1): pp 1–13, 2004.

[198] Lee Dong-Wook, Ban Chang-Bong, Sim Kwee-Bo, Seok Ho-Sik, Lee Kwang-Ju, Zhang Byoung-Tak, Behavior evolution of autonomous mobile robot using genetic programming based on evolvable hardware, Systems, Man, and Cybernetics, vol.5, pp 3835–3840, 2000.

[199] Lee Kwang-Ju, Zhang Byoung-Tak, Learning robot behaviors by evolving genetic programs, Industrial Electronics Society, vol.4, pp 2867- 872, 2000.

[200] Lee Wei-Po, An evolutionary system for automatic robot design Systems, Man and Cybernetics, IEEE Inter. Conference, Vol 4, pp 3477–3482, 1998.
[201] Leung KS, Lee KH, Cheang SM, Parallel programs are more evolvable than sequential programs, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 107-118, 2003.
[202] Levine DM, Stephan D, Krehbiel TC, Berenson ML, Statistics for Managers, 4th Edition, Pearson Prentice Hall, 2005.
[203] Lin JY, Ke HR, Chien BC, Yang WP, Designing a classifier by a layered multi-population genetic programming approach, Pattern Recognition 40 (8), pp 2211-2225, 2007.
[204] Liu Yi, Khoshgoftaar T. M., Genetic programming model for software quality classification", High Assurance Systems Engineering, pp 127–136, 2001.
[205] Lobo F G and Lima C F, A review of adaptive population sizing schemes in genetic algorithms. In GECCO05: Proceedings of the 2005 workshops on Genetic and evolutionary computation, pp 228-234, New York, USA 2005.
[206] Lobo F G and Lima C F, Revisiting evolutionary algorithms with on-the-fly population size adjustment, in GECCO05: Proceedings of the 2005 workshops on Genetic and evolutionary computations, pp228-234, New York, USA, 2005.
[207] Lones M A, Tyrrell A M, Enzyme genetic programming, Proceedings of the 2001 Congress on Evolutionary Computation, vol. 2, pp 1183 – 1190, 2001.
[208] Lu H and Yen G G, Dynamic Population size in multiobjective evolutionary algorithms, In Congress on Evolutionary Computation (GEC2002), vol. 2, pp 1648-1553, 2002.
[209] Luke S., Two fast tree-creation algorithms for genetic programming, IEEE Transactions on Evolutionary Computation 4 (2000) (3), pp 274–283, 2000.
[210] Luke S, Panait L, Fighting Bloat with Nonparametric Parsimony Pressure, Parallel Problem Solving from Nature - PPSN VII, Lecture Notes in Computer Science, Number 2439, pp 411, Springer Verlag, 2002.
[211] Luke S, Panait L, Lexicographic Parsimony Pressure, GECCO 2002, Proceedings of the Genetic and Evolutionary Computation Conference, pp 829-836, Springer Verlag, 2002.
[212] Luke S, Balan GC, Panait L, Population implosion in genetic programming, Genetic and Evolutionary Computation GECCO 2003, PT II, Proceedings Lecture Notes in Computer Science, pp 1729-1739, 2003.
[213] Luke S., Partait L., A comparison of bloat control methods for genetic programming, Evolutionary Computation, vol. 14 (3), pp 309-344, 2006.
[214] Majeed H. and Ryan C., A less destructive, context-aware crossover operator for GP. In P. Collet and et al, editors, Proceedings of EuroGP 2006, vol. 3905 of LNCS, pp 36–48. Springer-Verlag, 2006.
[215] Majeed H, Ryan C, Using context-aware crossover to improve the performance of GP, GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp 847-854, 2006.
[216] Majeed H, Ryan C, On the Constructiveness of Context-Aware Crossover, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1659-1666, 2007.
[217] Manrique D, Marquez F, Rios J, Rodriguez-Paton A, Grammar based crossover operator in genetic programming, Artificial Intelligence And Knowledge Engineering Applications: A Bioinspired Approach, PT 2, Proceedings Lecture Notes in Computer Science, pp 252-261, 2005.
[218] Manrique D, Ríos J, Rodríguez-Patón A, Evolutionary system for automatically constructing and adapting radial basis function networks. Int J Neuro-computation, pp1-16, 2006.
[219] Marchesi B., Stelle A. L., Lopes H. S., Detection of epileptic events using genetic programming, Engineering in Medicine and Biology society, vol.3, pp. 1198-1201, 1997.
[220] Marko K. A. and Hampo R. J, Application of genetic programming to control of vehicle systems, Intelligent Vehicles '92 Symposium, Proceedings of the, pp 191–195, 1992.
[221] Martin M. C., Genetic programming for real world robot vision, Intelligent Robots and System, vol.1, pp 67–72, 2002.
[222] Maxwell R. J., Pattern recognition analysis of 1H NMR spectra from human tumour biopsy extracts: a European union concerted action project, IEE Colloquium on Realising Clinical Potential of Magnetic Resonance Spectroscopy: The Role of Pattern Recognition (Ref. No: 1997/082), Pages: 2/1 - 2/3, 1997.
[223] McKay R. I. , Fitness sharing in genetic programming, in Proc. Genetic Evolutionary Computation Conf., D.Whitley et al., Eds., Las Vegas, NV, pp 435–442, 2000.
[224] McKay R., Abbass H.A., Anti-correlation: a diversity promoting mechanisms in ensemble learning, Austral. Journal. Intelligent. Information, Process. Systems (3/4), pp 139–149, 2001.
[225] McNutt G., Using co-evolution to produce robust robot control, Decision and Control, Vol 3, pp 2515–2520, 1997.

[226] McPhee NF and Miller JD, Accurate replication in genetic programming. In L. Eshelman, editor, Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), Morgan Kaufmann, pp 303–309, 1995.
[227] McPhee N. F. and Hopper N. J. , Analysis of genetic diversity through population history, in Proc. Genetic Evolutionary Computation Conf., W. Banzhaf et al., Eds., FL, pp 1112–1120, 1999.
[228] McPhee NF, Jarvis A, Crane EF, On the strength of size limits in linear genetic programming, Genetic and Evolutionary Computation GECCO 2004, PT 2, Proceedings Lecture Notes in Computer Science, pp 593-604, 2004.
[229] Messom C. H, Walker M. G., Evolving cooperative robotic behaviour using distributed genetic programming, Control, Automation, Robotics and Vision, vol.1, pp 215–219, 2002.
[230] Miller JF, Thomson P, Cartesian genetic programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 121-132, 2000.
[231] Miller JF, Smith SL, Redundancy and computational efficiency in Cartesian genetic programming , IEEE Transactions on Evolutionary Computation 10 (2), pp 167-174, 2006.
[232] Monsieurs P, Flerackers E, Reducing bloat in genetic programming, Computational Intelligence: Theory and Applications, Proceedings Lecture Notes in Computer Science, pp 471-478, 2001.
[233] Moore F W, Garcia O N, A new methodology for reducing brittleness in genetic programming, Aerospace and Electronics Conference, NAECON, Proceedings of the IEEE, vol.2, pp 757 – 763, 1997.
[234] Muntean O, Diosan L, Oltean M, Best SubTree genetic programming, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1667-1673, 2007.
[235] Nanduri DT, Ciesielski V, Comparison of the effectiveness of decimation and automatically defined functions. Knowledge-based Intelligent Information and Engineering Systems, PT 3, Proceedings Lecture Notes in Artificial Intelligence, pp 540-546, 2005.
[236] Nguyen X H, McKay R I, Essam D, Representation and structural difficulty in genetic programming, IEEE Transactions on Evolutionary Computation, vol. 10, Issue 2, pp 157-166, 2006.
[237] Niehaus J, Banzhaf W, Adaption of operator probabilities in genetic programming, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 325-336, 2001.
[238] Niehaus J, Igel C, Banzhaf W, Reducing the number of fitness evaluations in graph genetic programming using a canonical graph indexed database, Evolutionary Computation 15 (2): pp 199-221, 2007.
[239] Niimi A, Tazaki E, Extended genetic programming using reinforcement learning operation, IEEE International Conference on Systems, Man, and Cybernetics, SMC '99 Conference Proceedings, vol.5, pp 596–600, 1999.
[240] Niwa T and Iba H, Distributed Genetic Programming - Empirical Study and Analysis, In J.R. Koza, D. Goldberg, D.B. Fogel, and R.L. Riolo (editors), Genetic Programming 1996: Proceedings of the First Annual Conference, pp. 339-344, 38-31 July, Stanford University, CA: MIT Press, 1996.
[241] Nordin J P, Advances in Genetic Programming, K. E. Kinnear Jr., Ed. Cambridge, MA: MIT Press, vol. 1, ch. Chapter 14, pp. 311–331, 1994.
[242] Nordin P. and Banzhaf W., Complexity compression and evolution, in Proc. 6th Int. Conf. Genet. Algorithms, Pittsburgh 1995 (ICGA-95), L. Eshelman (ed.), Morgan Kaufmann: San Francisco, pp 310–317, 1995.
[243] Nordin P., Francone F., and Banzhaf W., Explicitly defined introns and destructive crossover in genetic programming, in Advances in Genetic Programming, K. E. Kinnear, Jr, and P. J. Angeline (eds.), MIT Press: Cambridge, MA, vol. 2, pp 111–134, 1995.
[244] Nordin J P., Banzhaf W., and Francone F. D., Advances in Genetic Programming, Spector L., Langdon W. B., O'Reilly U.-M., and Angeline P. J., Eds. Cambridge, MA: MIT Press, 1999, vol. 3, ch. 12, pp 275–299, 1999.
[245] Nordin P, Hoffmann F, Francone F D, Brameier M, Banzhaf W, AIM-GP and parallelism, Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, vol. 2, pp 1059-1066, 1999.
[246] O'Reilly U M, Oppacher F, Hybridized crossover-based search techniques for program discovery, IEEE International Conference on Evolutionary Computation, vol.2, pp 573-578, 1995.
[247] O'Reilly U M, Oppacher F, The troubling aspects of a building block hypothesis for genetic programming, In Whitely and Vose, editors, Foundation of Genetic Algorithms 3, pp 73-88, 1995.
[248] Ok S., Miyashita K., Hase K., Evolving bipedal locomotion with genetic programming – a preliminary report", Evolutionary Computation, vol. 2, pp 1025–1032, 2001.
[249] Oltean M, Solving even-parity problems using traceless genetic programming, Congress on Evolutionary Computation. CEC2004, vol.2 2, pp 1813–1819, 2004.

[250] Oussaidène M, Chopard B, Pictet O V, Tomassini M, Parallel genetic programming: An application to trading models evolution, in Genetic Programming 1996: Proceedings of the First Annual Conference, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, pp. 357–380, 1996.

[251] Page J, Poli R, Langdon WB, Smooth uniform crossover with smooth point mutation in Genetic Programming: A preliminary study, Genetic Programming Lecture Notes in Computer Science, pp 39-48, 1999.

[252] Panait L, Luke S, Alternative bloat control methods, Genetic and Evolutionary Computation GECCO 2004, PT 2, Proceedings Lecture Notes in Computer Science, pp 630-641, 2004.

[253] Perkis T, Stack-Based Genetic Programming, Proceedings of the 1994 IEEE World Congress on Computational Intelligence, vol. 1, pp 148-153, 1994.

[254] Perry J E, The effect of population enrichment in genetic programming, Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp 456 - 461 vol.1, 1994.

[255] Platel MD, Clergue M, Collard P, Size control with maximum homologous crossover, Artificial Evolution Lecture Notes in Computer Science, pp 13-24, 2006.

[256] Poli R. and Langdon W. B., A new schema theory for genetic programming with one-point crossover and point mutation, in Genetic Programming 1997: Proceedings of the Second Annual Conference (J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds.), (Stanford University, CA, USA), pp. 278–285, Morgan Kaufmann, 1997.

[257] Poli R. and Langdon W. B., Schema theory for genetic programming with one-point crossover and point mutation, Evolutionary Computation, vol. 6, no. 3, pp 231–252, 1998

[258] Poli R, A simple but theoretically-motivated method to control bloat in genetic programming, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 204-217, 2003.

[259] Poli R and Langdon WB, On the search properties of different crossover operators in genetic programming. In Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, pp 293–301, 1998.

[260] Poli R., Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In Alden et al, Schmitt editors, Foundations of Genetic Algorithms: 8th International Workshop (FOGA), pp 132-155, 2005.

[261] Punch W.F., Zongker D., Goodman E.D., The royal tree problem, a benchmark for single and multi-population genetic programming, in: P.J. Angeline, K.E. Kinnear, Jr. (Eds.), Advances in Genetic Programming vol. 2, The MIT Press, Cambridge, MA, USA, 1996, pp 299–316 (Chapter 15).

[262] Punch W F, How effective are multiple populations in genetic programming," in Proceedings of the Third Annual Conference on Genetic Programming, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds. San Mateo, CA: Morgan Kaufmann, pp 308–313, 1998.

[263] Rechenberg I, Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart, 1973.

[264] Riccardo Poli, Exact Schema Theory for Genetic Programming and Variable-Length Genetic Algorithms with One-Point Crossover", Genetic Programming and Evolvable Machines, 2, 123–163, © 2001 Kluwer Academic Publishers, 2001.

[265] Ritchie M.D., White B.C., Parker J.S., Hahn L.W. and Moore J.H., Optimization of neural network architecture using genetic programming improves detection of gene–gene interactions in studies of human diseases, BMC Bioinformatics 4 (2003), p 28, 2003.

[266] Ritchie MD, Coffey CS, Moore JH, Genetic programming neural networks as a bioinformatics tool for human genetics Genetic and Evolutionary Computation - GECCO 2004, PT 1, Proceedings Lecture Notes in Computer Science, pp438-448, Part 1 2004.

[267] Rizki M. M., Zmuda M. A. , Tamburino L. A., Evolving pattern recognition systems, Evolutionary Computation, pp 594–609, 2002.

[268] Roberts SC, Howard D, Koza JR, Evolving modules in genetic programming by subtree encapsulation, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 160-175, 2001.

[269] Rochat D, Tomassini M, Vanneschi L, Dynamic size populations in distributed genetic programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 50-61, 2005.

[270] Rodriguez-Vazquez K, Fonseca CM, Fleming PJ, Identifying the structure of non-linear dynamic systems using multiobjective genetic programming. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, pp 531–547, 2004.

[271] Rosca J P. and Ballard D H, Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, pp 1-30, February 1994.

[272] Rosca J P, Genetic programming exploratory power and the discovery of functions," in Proc. 4th Conf. Evolutionary Programming, J. R.McDonnell et al., Eds.,pp 719–736, 1995.

[273] Rosca J. P., Entropy-driven adaptive representation, in Proc. Workshop Genetic Programming: From Theory to Real-World Applications, J. Rosca, Ed., Tahoe City, CA, pp 23–32, 1995.

[274] Rosca J. P, Analysis of complexity drift in genetic programming, In Koza et. al, editors, Genetic Programming, Proceedings of the second annual conference, pp 286-294, 1997.

[275] Ryan C., Pygmies and civil servants, in Advances in Genetic Programming, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press, ch. 11, pp 243–263, 1994.

[276] Ryan C, Collins J J, O'Neill M. Grammatical evolution: Evolving programs for an arbitrary language, In W. Banzhaf et al, 1st European Workshop on Genetic Programming, vol. 1391 of Lecture Notes in Computer Science. Springer, 1998.

[277] Salhi A, Glaser H, De Roure D, Parallel implementation of a genetic-programming based tool for symbolic regression, Inform. Processing Lett., vol. 66, no. 6, pp. 299–307, 1998.

[278] Sanchez L, Interval-valued GA-P algorithms, IEEE Transactions on Evolutionary Computation, vol. 4 , Issue 1, pp 64-72, 2000.

[279] Schwefel H. P., Evolution and Optimum Seeking, Wiley, New York, 1995.

[280] Searson D. P., Willis M. J. and Montague G. A., Evolutionary design of process Controllers, International Conference on Control, pp1 456–1461, 1998.

[281] Seehuus R., Protein motif discovery with linear genetic programming, Knowledge-based Intelligent Information and Engineering Systems, PT 3, Proceedings Lecture Notes in Artificial Intelligence, vol. 3683, pp 770-776, 2005.

[282] Sharman K. C., Alcazar A. I. E., Li Y., Evolving signal processing algorithms by genetic Programming, Genetic Algorithms in Engineering Systems: Innovations and Applications, pp. 473–480, 1995.

[283] Silva S and Almeida J S, Dynamic maximum tree depth - a simple technique for avoiding bloat in tree-based GP. In E. Cantu-Paz, J. A. Foster, K. Deb, et al., editors, GECCO-2003, LNCS, Chicago, IL, USA, Springer, 2003.

[284] Silva S and Costa E, Dynamic limits for bloat control - variations on size and depth. In K. Deb, R. Poli, W. Banzhaf, et al., editors, GECCO-2004, LNCS, pp 666-677, Seattle, WA, USA, Springer, 2004.

[285] Silva S, Silva P J N, Costa E. Resource-limited genetic programming: Replacing tree depth limits. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, et al., editors, ICANNGA-2005, pages 243-246, Coimbra, Portugal, Springer, 2005.

[286] Silva S and Costa E, Comparing tree depth limits and resource-limited GP, The 2005 IEEE Congress on Evolutionary Computation, vol. 1, pp 920 – 927, 2005.

[287] Silva S and Costa E, Resource-limited genetic programming: the dynamic approach, Genetic And Evolutionary Computation Conference, Proceedings of the 2005 conference on Genetic and evolutionary computation, pp 1673-1680 , 2005.

[288] Soule T., Foster J. A., Dickinson J., Code growth in genetic programming. In Genetic Programming: Proceedings of the First Annual Conference, (Koza, J. R., Goldberg, D. E., Fogel, D. B. & Riolo, R. L., eds), MIT Press, Stanford University, CA, USA, pp. 215-223,1996.

[289] Soule T and Foster J A, Effects of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation, 6(4), pp 293-309, 1999.

[290] Soule T, Exons and code growth in genetic programming. In J. A. F. et al., editor, EuroGP 2002, volume 2278 of LNCS,Springer-Verlag, pp 142–151, 2002.

[291] Soule T and Heckendorn RB, An analysis of the causes of code growth in genetic programming, Genetic Programming and Evolvable Machines, 3, pp 283–309, 2002.

[292] Spinosa E, Pozo A, Controlling the population size in genetic programming, Advances in Artificial Intelligence, Proceedings Lecture Notes in Artificial Intelligence, pp 345-354, 2002.

[293] Steele Guy L. Jr., Common LISP, Second edition, Digital Press, 1990.

[294] Stevens J, Heckendorn RB, Soule T, Exploiting disruption aversion to control code bloat, GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp 1605-1612, 2005.

[295] Stone P and Veloso M, Layered Learning. In Proc. 17th International Conf. on Machine Learning, Springer-Verlag, pp 369-381, 2000.

[296] Streeter MJ, The root causes of code growth in genetic programming, Genetic Programming, Proceedings Lecture Notes in Computer Science, pp 443-454, 2003.

[297] Svangard N, Nordin P, Lloyd S, Using genetic programming with negative parsimony pressure on exons for portfolio optimization, The 2003 Congress on Evolutionary Computation, CEC '03, vol.2, pp 1014–1017, 2003.

[298] Tackett W A, Recombination, selection and the genetic construction of computer programs, Ph.D. dissertation, University of Southern California, Department of Electrical Engineering Systems, 1994.

[299] Tackett W. A., Carmi A., The unique implications of brood selection for genetic programming, IEEE World Congress on Computational Intelligence, vol.1, pp 160–165, 1994.

[300] Tan K C, Lee T H and Khor E F, Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization, IEEE-EC, vol. 5, pp 565-588, 2001.

[301] Tanev I, Uozumi T, Ono K, Parallel genetic programming: component object-based distributed collaborative approach, 15th International Conference on Information Networking, Proceedings , pp 129–136, 2001.

[302] Teller A., Veloso M., Algorithm evolution for face recognition: what makes a picture difficult, Evolutionary Computation, vol.2, pp 608-613, 1995.

[303] Terrio M.D., Heywood M. I., Directing crossover for reduction of bloat in GP, Electrical and Computer Engineering, vol. 2, pp 1111–1115, 2002.

[304] Tomassini M, Parallel and distributed evolutionary algorithms: A review, in Evolutionary Algorithms in Engineering and Computer Science, P. Neittaanmki, K. Miettinen, M. Mkel, and J. Periaux, Eds. Chichester, U.K.: Wiley, 1999.

[305] Tomassini M, Vanneschi L, Fernandez F, Galeano G, A study of diversity in multipopulation genetic programming, Artificial Evolution Lecture Notes in Computer Science, pp 243-255, 2004.

[306] Tsakonas A., A comparison of classification accuracy of four genetic programming evolved intelligent structures, Information Sciences, vol. 176 (6), pp 691-724, 2006.

[307] Tunstel E., Akbarzadeh-T M.R., Kumbla K., Jamshidi M., Soft computing paradigms for learning fuzzy controllers with applications to robotics", Fuzzy Information Processing Society, pp 355–359, 1996.

[308] Turing A M, On computable numbers, with an application to the Entsheidungsproblem, Proc. London Math Soc, 42, pp 230-265, 1936.

[309] Uchibe E., Nakamura M., Asada M., Co-evolution for cooperative behavior acquisition in a multiple mobile robot environment, Intelligent Robots and Systems, vol.1, pp 425–430, 1998.

[310] Vonk E., Jain L. C., Veelenturf L. P. J., Johnson R., Automatic generation of a neural network architecture using evolutionary computation", Proceedings of Electronic Technology Directions to the Year 2000, pp 144–149, 1995.

[311] Wagner N and Michalewicz Z, Genetic programming with efficient population control for financial time series prediction. In E. D. Goodman, editor, GECCO-2001 Late Breaking Papers, San Francisco, CA, USA, pp 458-462 2001.

[312] Walker J A, Miller J F, The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming, IEEE Transactions on Evolutionary Computation (Accepted for future publication), 2007.

[313] Wang G, Soule T, How to choose appropriate function sets for genetic programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 198-207, 2004.

[314] Watanabe K., Izumi K., A survey of robotic control systems constructed by using evolutionary computations, Systems, Man, and Cybernetics, vol 2, pp 758 - 763, 1999.

[315] Whigham P.A., Grammatically-based genetic programming, in: J.P. Rosca, (Ed.), Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (Tahoe City, California, USA, 1995) pp33–41, 1995.

[316] Wieczorek W, Czech ZJ, Grammars in genetic programming, Control and Cybernetics 29 (4), pp 1019-1030, 2000.

[317] Wolpert D H and Macready W G, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, pp 67-82, 1997.

[318] Wong Man Leung, Leung Kwong Sak, An induction system that learns programs indifferent programming languages using genetic programming and logic grammars" Tools with Artificial Intelligence, pp 380–387, 1995.

[319] Wong ML, Leung KS, Applying logic grammars to induce sub-functions in genetic programming. Evolutionary Computation 2, pp737–740, 1995.

[320] Wong M L, Leung K S, Combining genetic programming and inductive logic programming using logic grammars, IEEE International Conference on Evolutionary Computation, vol.2, pp 733–736, 1995.

[321] Wong M L, Leung K S, Data Mining using Grammar based Genetic Programming and applications, Kluwer Academic, Boston, 2000.

[322] Wongseree W., Chaiyaratana N., Vichittumaros K., Winichagoon P., Fucharoen S., Thalassaemia classification by neural networks and genetic programming, Information Sciences, vol. 177 (3), pp 771-786, 2007.

[323] Woodward JR, Complexity and Cartesian Genetic Programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 260-269, 2006.
[324] Wyns B, Sette S, Boullart L, Self-improvement to control code growth in genetic programming, Artificial Evolution Lecture Notes in Computer Science, pp 256-266, 2004.
[325] Wyns B, Boullart L, De Smedt P J, Limiting code growth to improve robustness in tree-based genetic programming, GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1763, 2007.
[326] Xie HY, Diversity control in GP with ADF for regression tasks, AI 2005: Advances in Artificial Intelligence Lecture Notes in Artificial Intelligence, pp 1253-1257, 2005.
[327] Xie HY, Zhang, Andreae P, Automatic Selection Pressure Control in Genetic Programming, Sixth International Conference on Intelligent Systems Design and Applications, ISDA '06, Volume: 1 , pp 435–440, 2006.
[328] Xie HY, Zhang MJ, Andreae P, Population clustering in genetic programming, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 190-201, 2006.
[329] Xie Huayang, Zhang Mengjie, Andreae Peter, An analysis of constructive crossover and selection pressure in genetic programming, GECCO, pp 1739-1748, 2007.
[330] Xie Huayang, Zhang Mengjie, Andreae Peter, Another investigation on tournament selection: modelling and visualisation. GECCO, pp 1468-1475, 2007.
[331] Yanagiya M, Efficient genetic programming based on binary decision diagrams, IEEE International Conference on Evolutionary Computation, vol. 1. pp 234-239, 1995.
[332] Yuen CC, Selective crossover using gene dominance as an adaptive strategy for genetic programming. Msc intelligent systems, University College, London, UK, September 2004.
[333] Zhang B T, Muhlenbein H., Synthesis of sigma-pi neural networks by the breeder genetic programming, IEEE World Congress on Computational Intelligence, vol.1, pp 318-323, 1994.
[334] Zhang B T and Miihlenbein H, Balancing accuracy and parsimony in genetic programming. Evolutionary Computation, 3(1), pp 17-38, 1995.
[335] Zhang H, Lu YN, Wang F, Grammar based genetic programming using linear representations, Chinese Journal of Electronics 12 (1): pp 75-78, 2003.
[336] Zhang L, Nandi AK, Neutral offspring controlling operators in genetic programming, Pattern Recognition 40 (10), pp 2696-2705, Oct 2007.
[337] Zhang MJ, Gao XY, Lou WJ, Qian DP, Investigation of brood size in GP with brood recombination crossover for object recognition, PRICAI 2006: Trends in Artificial Intelligence, Proceedings Lecture Notes in Artificial Intelligence, pp 923-928, 2006.
[338] Zhang M, Gao X, Lou W, Looseness Controlled Crossover in GP for Object Recognition, IEEE Congress on Evolutionary Computation, CEC 2006, pp 1285-1292, 2006.
[339] Zhang MJ, Wong P, Qian DP, Online program simplification in genetic programming, source, Simulated Evolution and Learning, Proceedings Lecture Notes in Computer Science, pp 592-600, 2006.
[340] Zhang Y and Rockett PI. Evolving optimal feature extraction using multi-objective genetic programming: A methodology and preliminary study on edge detection. In Beyer et al., editors, Genetic and Evolutionary Computation Conference (GECCO 2005), pp 795–802, 2005.
[341] Zhang Y and Rockett PI, Feature extraction using multi-objective genetic programming. In Y. Jin, editor, Multi-Objective Machine Learning. Springer, Heidelberg, 2006.
[342] Zhang YQ, Chen HS, Improved approach of Genetic Programming and applications for data mining, Advances in Natural Computation, pt 1, Lecture Notes in Computer Science, pp 816-819, 2006.
[343] Zitzler E and ThieIe L, An Evolutionary Algorithm for Multiobjective optimization: The Strength Pareto Approach,, Swiss Federal Institute of Technology (ETH) Zurich, TIK-Report, No. 43, 1998.
[344] Zitzler E and Thiele L, Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionury Computation, 3(4), pp 257-27, 1999.
[345] Zitzler E, Laumanns M and Thiele L, SPEA2: Improving the Performance of the Strength Pareto Evolutionary Algorithm, Technical Report 103, Computer Engineering and Communication Networks Lab (TLK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.
[346] Zvada S, Vanyi B, Improving grammar-based evolutionary algorithms via attributed derivation trees, Genetic programming, Proceedings Lecture Notes in Computer Science, pp 208-219, 2004.

Index