An Investigation into Grammatical Evolution Search Strategies

John O'Sullivan

Department of Computer Science Information Systems

University of Limerick

A thesis presented for the award of the MSc degree of the

University of Limerick

Supervisor: Dr. Conor Ryan

Submitted to the University of Limerick March 2003

Contents

List of Figures

List of Tables

Acknowledgements

I must first thank my thesis adviser Dr. Conor Ryan for his steadfast support and patient guidance over the course of this research. I am also grateful to Dr. Michael O'Neill and J.J. Collins whose source code formed the basis for much of the early research.

This work was funded by my former employer Tellabs Ltd. I would also like to acknowledge the considerable encouragement and support of my current employer Homenet Communications Ltd.

Abstract

Grammatical Evolution (GE) is an Evolutionary Algorithm (EA) that can generate programs in an abritrary language. GE employs a genotype-phenotype mapping process which uses variable length binary strings to select productions from a Backus Naur Form grammar. To-date GE has used a Genetic Algorithm as the means of directing the search, however, its ability to separate the search space and solution space makes it suitable for use with a variety of search mechanisms.

In this research the performance and characteristics of GE are compared across a range of metaheuristics including, Random Search (RS), Hill Climbing (HC), Simulated Annealing (SA), Evolutionary Programming (EP), Evolutionary Strategies (ES) and Genetic Algorithms (GAs). The basis for this evaluation is a series of experimental trials in which GE attempts to solve a selection of benchmark problems from the field of Evolutionary Computing (EC).

We attempt to profile the problem search spaces revealing complex, noisy, discontinuous multimodal landscapes which prove difficult for the deterministic gradient following approaches of HC and SA. We see solutions to our selected problems exist as sharp isolated peaks of maximum fitness with some evidence from our use of RS suggesting that some of the problems have a high density of such peaks.

This research demonstrates GE's ability to work successfully and seamlessly with a broad range of different search approaches and identifies distinct performance advantages for population based approaches like EP, ES and GAs. This flexibility establishes GE as a powerful and flexible mechanism for the automatic generation of grammar based code and programs.

Chapter 1

Introduction to the Thesis

1.1 Introduction

Grammatical Evolution (GE) is an Evolutionary Algorithm (EA) that uses variable length binary strings to select productions from a Backus Naur Form grammar. A feature of Grammatical Evolution is the ability to separate the search space from the solution space. This aspect, which allows different methods to be used to explore the search space, is examined in this research by evaluating the performance and characteristics of GE using a diverse range of search methods (Metaheuristics) on a selection of benchmark problems from the field of *Evolutionary Computation (EC)*.

1.2 Contributions of the Thesis

The principal contribution of the thesis is the evaluation of GE with a range of metaheuristics. In an effort to better understand the nature of the search space presented by the GE process these metaheuristics have been evaluated on a range of traditional Genetic Programming (GP) benchmark problems.

This is the first time that research on GE has focused on using different search strategies, up to this point GE had been exclusively used with a Variable-Length Genetic Algorithm as the search engine.

The work brings together a useful practical summary of many of the key metaheuristics in use today, providing a step-by-step guide to their underlying algorithms and operators. The research has also resulted in the generation of a significant body of object orientated code implementing all of the selected metaheuristics. By using Random search as one of the search mechanisms in this research we provide some interesting insights into the relative difficulty of the selected problems.

1.3 Organisation of the Thesis

Chapter two of the thesis provides an introduction to GE, providing an insight into the key workings of the technique. It provides some background on the specification of grammars, introduces the Backus Naur Form of grammar and illustrates the mapping process used by GE in evolving programs from a grammar.

In chapter three we discuss Metaheuristics, providing an overview of the subject before dealing in detail with each of the metaheuristics featured in this research.

The selected problem set consisting of *The Santa Fe Trail problem*, *Symbolic Integration Symbolic Regression* and *Block Stacking* is introduced in chapter four, where we provide a description of each problem and presents its associated grammar.

Chapters seven through to ten focus on the results from experimental trials using the selected Metaheuristics, each chapter follows a similar structure

presenting the overall results and then attempting to analyse and improve the performance of the Metaheuristic under consideration.

The focus of chapter eleven is a comparative overview of all of the results provided in the previous chapters looking at relative performance with some discussion on the characteristics that the various Metaheuristics have in common and a contrast of those aspects that differentiate them.

Finally, chapter twelve summarises all of the key findings and identifies areas for future research.

1.4 Terminology

In the context of this thesis the word *Solutions* means a potential solution to a problem rather than the correct solution to the problem. A solution that actually solves the problem will be referred to as a *correct solution* or a *successful solution*.

All results have been tested for significance using a *Chi Square* significance test at a 98% confidence level. This test compares the difference between the actual frequencies (the observed frequencies in the data) with those which one would expect if no factor other than chance had been operating (the expected frequencies). The closer these two results are to each other, the greater the probablity that the observed frequencies are influenced by chance alone. When the text refers to changes as being significant or not significant it is within the context of this test.

Chapter 2

Grammatical Evolution

2.1 Introduction

This chapter introduces the main elements of Grammatical Evolution [22] [31]. It begins by providing an overview of grammars, in particular, focusing on the Backus Naur form used by GE. It also provides brief descriptions of derivation trees and parse trees, two important conceptual tools used to describe the application and structure of a grammar. The basic mechanism of GE is also introduced showing a step-by-step evolution of a solution using a simple grammar. Many of the terms associated with GE and referenced in the following chapters are also introduced here.

2.2 Grammars

Grammars provide a means of formally describing a language. Grammars consist of rules, which determine the linguistic structure of sentences in the language, terminals, which are the words or symbols of the language, and non-terminals, which expand into one or more terminals. In a *context free*

```
N = {expr,op}
T = {+,-,/,*,X,(,)}
S = <expr>
P is a set of rules:
<expr>     :: == <var> | <expr><op><expr> | <pre-op>(<expr>)
                 | (<expr>)
<var>      :: == X
<op>       :: == + | - | * | /
<pre-op>   :: == sin | cos | tan | log
```

Figure 2.1: Example Grammar presented in Backus Naur Form.

grammar (CFG) the syntax of a symbol, terminal or non-terminal is the same regardless of what other symbols surround it. Where there is a single non-terminal the grammar can be described as *closed* [10].

Backus Naur Form (BNF), an example of which is shown in Figure 2.1, is a convenient descriptive notation for context free grammars is a notation for expressing the grammar of a language in the form of production rules. BNF consists of *terminals*, which are items that can appear in the language, e.g. + - * etc. and *non-terminals*, which can be expanded into one or more terminals and non-terminals. A grammar can be represented by the tuple $< N, T, P, S >$, where N is the set of non-terminals, T the set of terminals, P a set of production rules used to map elements of N to T. S a start symbol is a member of N.

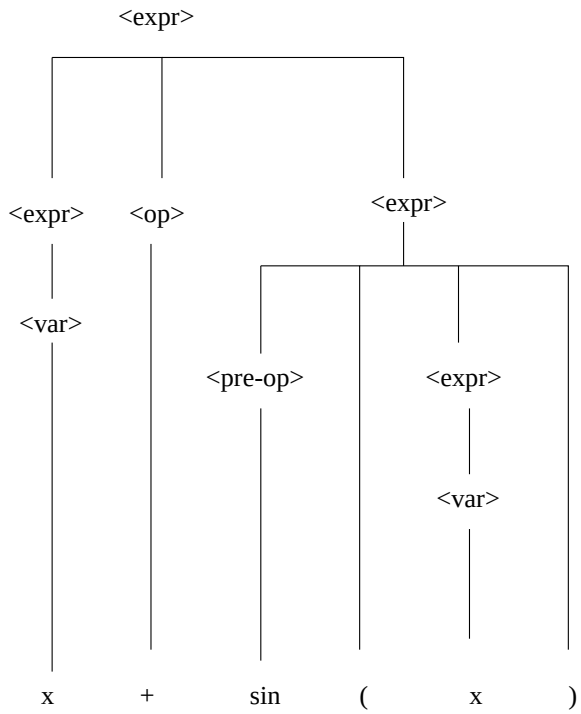

Figure 2.2: Derivation Tree for the Expression $x + sin(x)$.

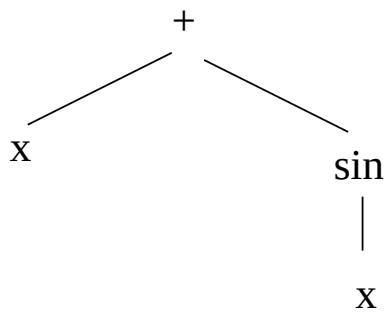

Figure 2.3: Parse Tree for the Expression $x + sin(x)$.

2.2.1 Derivation Trees

A derivation tree is a way of showing graphically the derivation of a sentence from a grammar. The generation of an expression in GE can be represented in this form. Figure 2.2 shows a derivation tree for the expression $x + sin(x)$ using the simple grammar shown above. At the top of this tree we see the first derivation step in which the non-terminal $< expr >$ is replaced by $< expr >< op >< expr >$.

2.2.2 Parse Trees

A parse tree concerns itself only with terminals. It assists the breakdown of the linguistic structure of the language. The parse tree for the expression $x + sin(x)$ is shown in Figure 2.3.

2.3 A Description of the Method

Grammatical Evolution (GE) [22] [31] is an evolutionary algorithm that can evolve complete programs in an arbitrary language using a variable-length binary string. The binary string is decoded into a genome, which in turn is used to select production rules from a Backus Naur Form (BNF) grammar

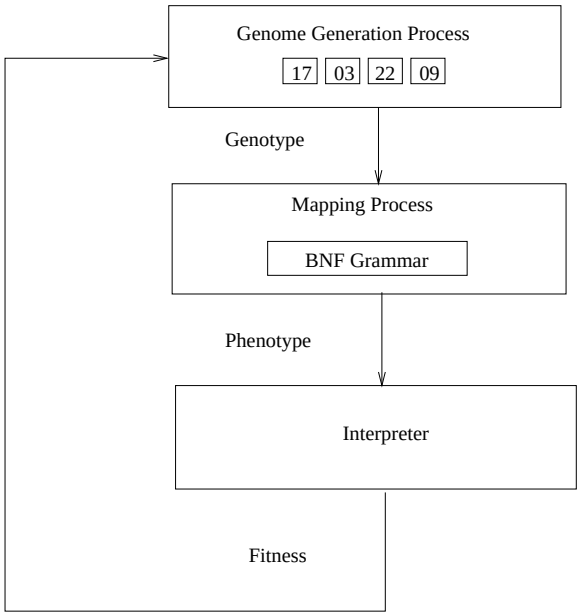

Figure 2.4: Grammatical Evolution, any metaheuristic capable of generating a variable length genome can be used with the system.

definition. The system, illustrated in Figure 2.4, can use any search mechanism capable of generating a variable length sequence of integer codons. The codons, typically in the range 0 - 255, are used to select productions from the target grammar using the following function:

Rule = (Codon value) MOD(Number of Rules for the current non-terminal)

A number of different codon values in the range 0 - 255 can cause selection of the same production rule. This occurs because the number of production rules associated with any non-terminal is generally small relative to the 256 expressible by a codon. This *Genetic Code Degeneracy* [22] permits *neutral mutation* which allows subtle changes in the search space without necessarily impacting the solution space (see Section 2.5).

Wrapping is a technique employed by GE which allows the mapping process re-use codons from the genome. Once all codons in a genome have been

used the process simply wraps to the start of the genome and continues the mapping process helping to improve the probability of a complete mapping of individuals onto programs.

The fitness associated with a solution is provided by evaluating the phenotype (i.e.,the program) using an interpreter for the target problem class. This fitness value is available to the generation mechanism to influence the future direction of the search strategy. The following example illustrates the process of GE for the following simple expression:

$$f(x) = sin(x) + x^2 + x$$

A grammar, which can be used to derive this simple expression, is shown below.

```
<expr>   :: == <var> | <expr><op><expr> | <pre-op>(<expr>)
                     | (<expr>)
<var>    :: == X
<op>     :: == + | - | * | /
<pre-op> :: == sin | cos | tan | log
```

We present this grammar in tabular form in Figure 2.1 and show the mapping process that occurs once the search engine has presented a codon to GE.

GE can use a depth first or a breadth first approach when mapping the codon string to an expression. The example shown in Table 2.2 and Table 2.3 illustrates the depth first approach which employs expansion of the left most non-terminal as the expression is created. Detailed commentary has been provided on the first nine derivation steps. The expanding non-terminal is

Non-Terminals		Productions			
		0	1	2	3
1.	<expr>	<var>	<expr><op><expr>	<pre-op>(<expr>	(<expr>)
2.	<var>	X			
3.	<op>	+	-	*	/
4.	<pre-op>	sin	cos	tan	log

Table 2.1: Simple grammar presented in table form.

shown in bold type through out Table 2.2 and Table 2.3 while the genome string used is shown in Figure 2.5. Depth first expansion has been used in all of the trials reported in later sections.

11	57	91	148	22	129	53	123	123	148	132	168	84	38	232

Figure 2.5: Genome used in example of mapping process.

2.4 Expressed and Unexpressed Codons

A feature of GE is the ability to use only those codons which are required to effect a successful mapping. The genome string presented to GE could, for example, contain 30 codons, if only 15 of these are required to map a successful solution then the remaining 15 remain unexpressed. Though these codons are unexpressed they are not completely redundant, because the polymorphic nature of GE means that these codons could come in to play as that genome evolves through mutation or recombination.

Step	Expression
0	$< expr >$ Start symbol.
1	$(<$ **expr** $>)$ 11 mod 4 = 3, Production 3 from row 1 was chosen.
2	$(<$ **expr** $>< op >< expr >)$ 57 mod 4 = 1, Production 1 from row 1 was chosen.
3	$((<$ **expr** $>) < op >< expr >)$ 91 mod 4 = 3, Production 3 from row 1 was chosen.
4	$((<$ **var** $>) < op >< expr >)$ 148 mod 4 = 0, Production 0 from row 1 was chosen.
5	$((X)<$ **op** $>< expr >)$ There is only one choice for $< var >$ so no codon was used for the transition to X.
6	$((X) * <$ **expr** $>)$ 22 mod 4 = 2, Production 2 from row 4 was chosen.
7	$((X) * <$ **expr** $>< op >< expr >)$ 129 mod 4 = 1, Production 1 from row 1 was chosen.
8	$((X) * <$ **expr** $>< op >< expr >< op >< expr >)$ 53 mod 4 = 1, Production 1 from row 1 was chosen.

Table 2.2: Analysis of Depth First Mapping.

Step	Expression
9	$((X) * (<\mathbf{expr}>) < op >< expr >< op >< expr >)$
10	$((X) * ((<\mathbf{expr}>)) < op >< expr >< op >< expr >)$
11	$((X) * ((< var >))<\mathbf{op}> < expr >< op >< expr >)$
12	$((X) * ((X))<\mathbf{op}> < expr >< op >< expr >)$
13	$((X) * ((X)) + <\mathbf{expr}> < op >< expr >)$
14	$((X) * ((X)) + <\mathbf{var}> < op >< expr >)$
15	$((X) * ((X)) + X<\mathbf{op}> < expr >)$
16	$((X) * ((X)) + X + <\mathbf{expr}>)$
17	$((X) * ((X)) + X + <\mathbf{pre-op}>(< expr >))$
18	$((X) * ((X)) + X + Sin(<\mathbf{expr}>))$
19	$((X) * ((X)) + X + Sin(<\mathbf{var}>))$
20	$((X) * ((X)) + X + Sin(X))$

Table 2.3: Analysis of Depth First Mapping (continued).

2.5 Genetic Code Degeneracy and neutral mutation

Genetic code degeneracy is a phenomenon that can be observed in the genetic code of biological organisms [5]. It occurs where there is redundancy in the coding required to specify aspects of the DNA. This redundancy can neutralise the effects of mutation in cases where the mutated bits are those that are redundant to the coding. Kimura's neutral theory of evolution [12] suggests that it is *neutral mutations* that are responsible for the genetic diversity found in natural populations.

Genetic code degeneracy occurs in GE by virtue of the fact that even though a codon can represent 256 distinct integer values, many of these values represent the same production rule. For example using the *MOD rule* with 4 productions would result in codon values 8, 4, 12 and 16 all selecting the same production 0. It is apparent from this that genomes with differing codon compositions can map to the same solution. This property of GE, which permits *neutral mutation* on redundant codings, allows subtle changes in the search space without necessarily impacting the solution space.

2.6 Summary

This chapter has introduced Grammatical Evolution, showing in detail how a variable length string of integer values can be used to derive an expression in any language given a suitable grammar. An introduction was provided to the subject of language grammars with particular focus on the Backus Naur Form. The concepts of genetic code degeneracy, neutral mutation, and expressed and unexpressed genes were also introduced.

Chapter 3

Metaheuristics

3.1 Introduction

This chapter provides an insight into the workings of the various metaheuristics used in this research. We first define what we mean by the term *metaheuristics* and then discuss each of the selected methods in turn. Random Search, Hill Climbing, Simulated Annealing, Evolutionary Programming, Genetic Programming, Evolutionary Strategies and Genetic Algorithms are examined in detail, providing background on the origins of the method, a short summary of the algorithm, and finally, some discussion on the operators used and the selection methods employed

Metaheuristics are a class of approximate methods that are designed to attack hard combinatorial optimzation problems. They are often employed where classical heuristics have failed to be effective and efficient. Metaheuristics are computational procedures rather than computational algorithms[1],

[1]An Algorithm is a specific class of computational procedure that consists of a well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time.

typically exploring the solution landscape by starting from some randomly chosen point. They differ from heuristics and from formal computational algorithms in that they assume very little knowledge of the problem domain. They are what Artificial Intelligence has termed weak methods in that they are general strategies intended to be applicable to entire classes of problems.

3.2 Genetic Algorithm

A Genetic Algorithm (GA) [9] [8] is a population-based method that simulates the evolution of individual structures through the process of selection, reproduction, recombination and mutation. The GA models itself on the processes of evolution in nature by creating a population of individuals represented by chromosomes. These are, in essence, a set of character strings that are analogous to the chromosomes that we see in our own DNA. The individuals in this population evolve over the lifetime of the algorithm.

Crossover or recombination, which is the process of exchanging genetic information, happens in an environment where the probability of selection is a function of the fitness of the individual. Fitness in this context is a measure of the individuals success in solving the problem. Individuals which solve the problem are said to be perfectly fit, they have achieved the highest possible score on the task. The fitness measure used to select individuals to undergo genetic operations such as crossover can range from simple fitness-proportionate selection to tournament-based selection where individuals in a subgroup compete and the fittest is selected. Mutation also plays a role in the GA process, while mutation may be random the process of the GA (as a simulation of a genetic process) itself is not a "random search" for a solution to a problem (highly fit individual). The GA uses stochastic processes, but

the result is distinctly non-random.

Previous analysis of GE [22] has used a Variable-Length GA as the means of directing the search. As in this previous analysis the experiments in this thesis use a steady-state population employing the standard GA operators of crossover and mutation. The GA used in this research employs variable-length one point crossover with random mutation (see Figure 3.1) operating at the bit-level level. Two non-standard operators, *Duplication* and *Swapping* (see Figure 3.2) which work at the codon level are also used. Duplication involves copying a sequence of genes of random length and placing them between the last and second last gene, while the Swap operator does a positional swap of two randomly selected genes.

The GA algorithm used in this research takes the following form:

1. Randomly create an initial population of n solutions.
2. Calculate and normalise the fitness of the individuals through scaling.
3. Select two parents using roulette wheel selection mechanism.
4. Create two offspring from the selected parents by applying crossover, mutation, duplication and swap operations at set probabilities.
5. Replace the least fit individual in the population with the selected offspring if they have lower fitness than the selected offspring.
6. Go to step 3 until n selections of offspring have been made.
7. Go to step 2 until the required solution has been found or the maximum generations have been reached.

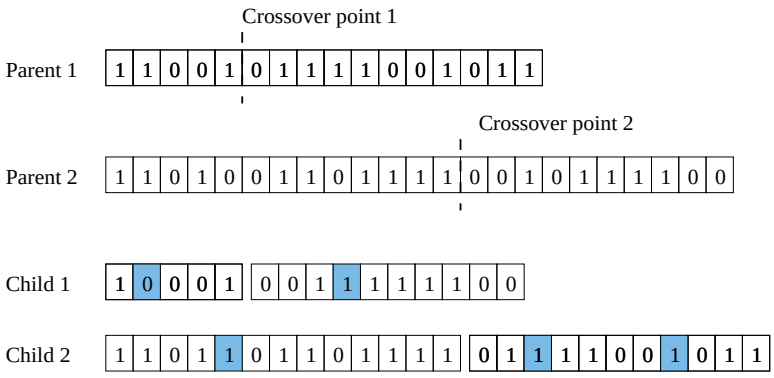

Figure 3.1: Crossover Operator with Mutation, used by GA. The figure shows the effects of the combined operators on selected genomes

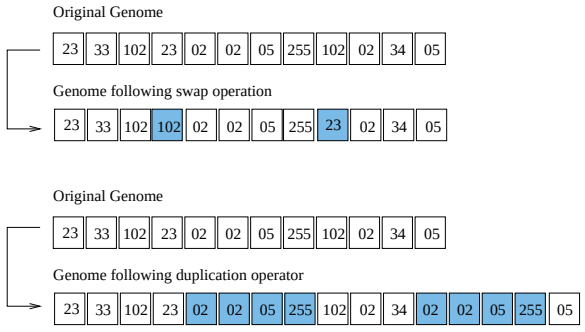

Figure 3.2: Duplicate and Swap Operators, used by GA. The figure shows the effects of each operator on a selected genome

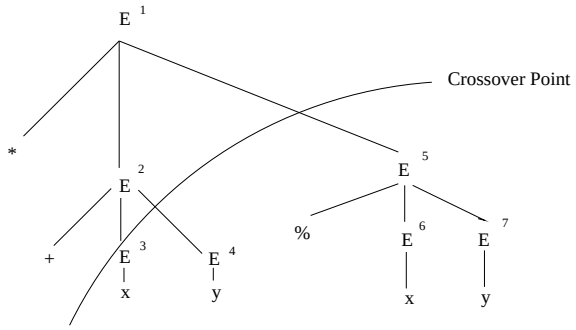

Figure 3.3: Ripple Crossover, Derivation Tree with Crossover site marked.

3.2.1 Ripple Crossover

Ripple Crossover [10] is a term used to describe the operation of crossover in GE. When a crossover point is selected in GE on a closed grammar (see Section 2.2) all sub trees to the right of the crossover point are removed leaving an incomplete spine with multiple crossover points. For a context free grammar with multiple symbols the sub-trees removed do not have a constant interpretation, often causing radical re-interpretation of the codons involved. One of the consequences of ripple crossover is that on average half of the genetic material of an individual is exchanged during the crossover operation creating a global search throughout the course of a run. The disruptive nature of this global search mitigates against getting stuck in local optima by driving improved fitness generation after generation.

The grammar used in this example is:

```
E ::== X | Y | (+ E E) | (* E E) | (- E E) | (/ E E).
```

Figure 3.3 shows the effect of crossover on the derivation tree for a closed grammar. All sub-trees to the right of the crossover point are removed leaving a spine (shown in Figure 3.4) and a number of tails (shown in Figure 3.5). The removal of crossover point 3 has also resulted in the removal of crossover

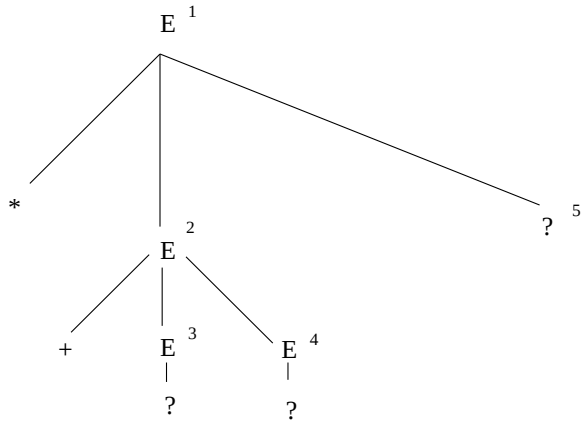

Figure 3.4: Ripple Crossover, Derivation Tree Spine after Crossover.

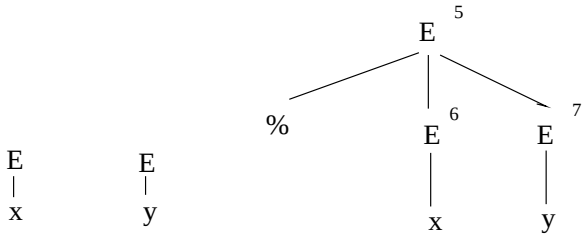

Figure 3.5: Ripple Crossover, Derivation Tree Tails after Crossover.

points 4, 5, 6 and 7, leaving an incomplete spine requiring 3 sub-trees to be complete. The removed tails, which are exchanged in the crossover operation, can then be used to complete crossover points in another tree.

3.3 Genetic Programming

Genetic Programming (GP) [14] [15] [16] is an Evolutionary Algorithm that evolves parse trees. Although it is not evaluated directly as part of this research it does provide a useful yardstick for performance on the problems evaluated in the trials detailed later. Much of the early analysis of GE [22] was presented as a comparative study against the efforts of GP on identical problems.

Parse trees (see section 2.2.2) can be used to represent programs in any language. These parse trees are evolved by GP using a combination of mutation and recombination.

Closure, which is an important property in GP, exists when any terminal integer or variable can be passed to any function and any return type of a function can be passed as an argument to any other function. This ensures legal trees are maintained as the parse trees change in the course of the evolution. However, this property also imposes the constraint that standard GP programs can only handle a single type.

The programs in the population are composed of elements from the FUNCTION SET and the TERMINAL SET, which are typically fixed sets of symbols selected to be appropriate to the solution of problems in the domain of interest. An example of these is shown below.

```
GP function set F = {+,*,-,/}
Arity of functions  {2,2,2,2}
GP terminal set T = {X,Y}
```

These are similar to the grammars shown in the previous section. The single non-terminal E ensures closure. In the Lisp programming language there is no distinction between the program and data, data objects can be manipulated as Lisp expressions, this feature, and its ability to handle the linked data structures used in GP has made it a popular choice in the development and implementation of the method. However the method is easily accommodated in many non-Lisp programming environments. Recombination in GP involves taking randomly selected sub trees in the individuals (selected according to fitness) and exchanging them. The basic algorithm is as follows:

1. Randomly create an initial population of random compositions of the functions and terminals of the problem (programs).
2. Evaluate each program in the population and assign it a fitness value.
3. Using fitness as the selection mechanism create a new population through mutation and recombination.
4. Go to step 2 until the required solution is found or the maximum evaluations have been reached.

3.4 Random Search

Random search cannot be strictly regarded as a metaheuristic, because it is essentially a random sampling of the search space. An initial solution is generated and its fitness calculated, then each new randomly generated solution is evaluated relative to the current one. A new solution is accepted if its fitness is greater than that of the current solution. This process continues for a defined number of evaluations. The basic random search algorithm used for these trials is shown below.

1. Randomly create an initial solution and determine its fitness.
2. Randomly create a new solution and determine its fitness.
3. Compute the change in fitness (δF) of the system between the two solutions.
4. If $\delta F > 0$ accept the new solution else reject the new solution and retain the current solution.
5. Go to step 2 until the required solution is found or the maximum evaluations have been reached.

3.5 Simulated Annealing

The Simulated Annealing search technique is modeled on the way crystals form in solids during the cooling process. The quality of the crystals formed in the cooling process is dependent on the rate of cooling. The randomness provided by thermal energy allows atoms to escape from locally optimal configurations (meta-stable states) and form an optimum minimum energy crystalline structure. The rate of cooling is governed by an annealing schedule, which consists of an initial temperature, a rule for decrementing the temperature, a set number of iterations at a particular temperature and a final temperature.

The process and algorithms used in simulated annealing comes from the laws of thermodynamics which state that at a temperature t, the probability P for a change in energy of magnitude δE is given by

$$P(\delta E) = e^{\frac{-\delta E}{kt}}$$

where k is the physical constant known as Boltzmann's constant. In a simulated version of annealing this equation is used within a system that is cooling toward a steady state. For a general optimization problem, the temperature is just a parameter that governs the probability of increasing the cost function at any step. The usual Metropolis algorithm form [20] used for general optimisation problems is

1. Select a starting temperature.
2. Randomly create an initial solution and determine its fitness.
3. Make a random trial change in the current solution.
4. Compute the change in fitness (δF) of the system due to the trial change.

5. If $\delta F > 0$ accept the new solution else compute the acceptance probability $p = e^{\frac{-\delta C}{t}}$.

6. Generate a random number r in the interval [0,1].

7. Accept the new solution if $p \geq r$ else retain the current solution.

8. Go to step 2 until the required solution has been found or the maximum evaluations have been reached.

9. Reduce the temperature, reset the evaluations counter to 0 and go to step 2 until the minimum temperature has been reached.

In simple terms the search technique permits the acceptance of dis-improving solutions with a probability which decreases as the process progresses and the temperature drops. The temperature must be decremented in sufficiently small steps, ensuring that the size of these steps has a logarithmic relationship with the number of iterations. A simple logarithmic cooling schedule is known to be optimal, allowing the temperature to be lowered at every iteration.

3.6 Hill Climbing

Hill Climbing is a simple but powerful metaheuristic. In its simplest form the search method attempts to find a global maximum by moving in an uphill direction. Random Mutation Hill-climbing (RMHC) [21] uses random sampling of nearby points in the search space in order to determine the uphill direction. The Hill Climbing algorithm used in these experiments uses a similar approach, however mutation is carried out at the codon level. A noted shortcomings of Hill Climbing is the possibility that it will wander randomly through the search space when presented with plateaus in which all neighboring points have similar fitness values or that it will become trapped when

it reaches a local minima/maxima. There are many variations on the basic algorithm, which attempt to address this weakness. One common variation is random-restart or multi-start hill climbing. Multi-start hill climbing [4] generates a random starting solution and then hill-climbs to a local optimum. Once at a local optimum it repeats this process by generating a new random solution from which to hill-climb. The algorithm used in the various experiments, which is based on the simple form of Hill Climbing, is shown below:

1. Randomly create an initial solution and determine its fitness.
2. Make a random trial change in the current solution.
3. Compute the change in fitness (δF) of the system between the two solutions.
4. If $\delta F > 0$ accept the new solution else reject the new solution and retain the current solution.
5. Go to step 2 until the required solution has been found or the maximum evaluations have been reached.

Both the Hill Climbing and Simulated Annealing algorithms used in this study use a selection of four operators to explore the search space. The operators have been selected to allow subtle changes in the genomes by allowing a single mutation select alternate productions from the BNF grammar. The operators used *Force-up* and *Force-down* (see Figure 3.6) provide a means of selecting alternate productions by incrementing or decrementing a single randomly chosen codon. Two additional operators *Grow* and *Shrink* provide a means of expanding and contracting the genome into different areas of the search space. The Force-up and Force-down operators will always force the

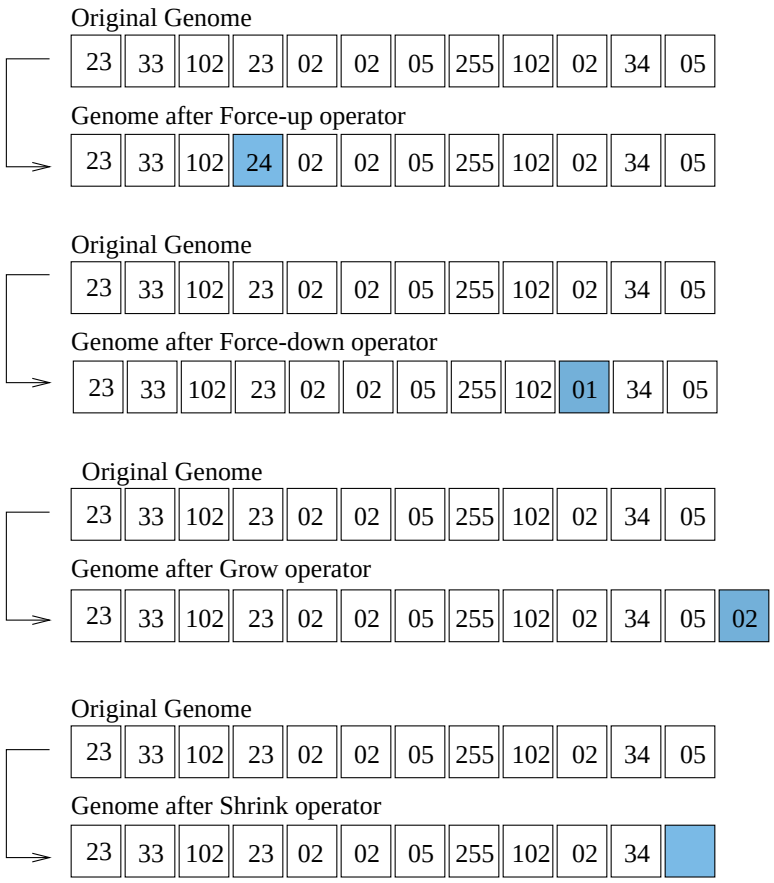

Figure 3.6: Operators, used by Hill Climbing and Simulated Annealing to explore the search space.

selection of a different production from the grammar, consequently *neutral mutation* discussed in section 2.5 does not occur. The shrink operator removes the last codon from the genome while the grow operator adds a single codon whose value is randomly chosen. One of the requirements of Simulated Annealing is that the applied mutations be probabilistically reversible, thus the probability of generating x from y needs to be equal to the probability of generating y from x. The implications of this are that the operator pairs Grow/Srink and Force-up/Force-down should have the same probability of occurrence.

3.7 Evolutionary Programming

Evolutionary Programming (EP) had its origins in the work of Fogel, Owens and Walsh in the 1960s. It re-emerged in the 1980s when his son L.J.Fogel extended the approach to applications involving continuous parameter optimisation. Standard EP in its original form does not use any self-adaption mechanism, however Meta-EP which has many features in common with Evolutionary Strategies (see section 3.8), incorporating variances into the genotype. These variances evolve along with the object values. Unlike Evolutionary Strategies Meta-EP does not use a recombination or crossover operator. Each parent produces one offspring through mutation. In recent years EP like ES has been more strongly associated with continuous parameter optimisation, however applications of EP with some form of self-adaption has also featured in discrete optimisations [3]. Both the continuous and discrete forms are described below.

The algorithm associated with the continuous form is shown below. In this context an individual or solution consists of a genome and one or more strategy variables. The mutation of the offspring includes the mutation of both the genome and the associated strategy variables.

1. Randomly create an initial population of n solutions.
2. Create n offsprings by making a copy of this population.
3. Mutate the offsprings.
4. Combine the original population and the mutated offsprings.
5. Determine the fitness of each member of the combined population.
6. Select the best n individuals from the combined population.

7. Go to step 2 until the required solution has been found or the maximum generations have been reached.

For continuous parameter optimisation using the meta-EP model mutation proceeds as follows: First the object variable is mutated as follows:

$$X_i' = X_i + \sqrt{V_i} \cdot N(0,1)$$

where

X_i' is the value of the mutated object variable at index i.

X_i is the current value of the object variable at index i.

$\sqrt{V_i}$ is the square root of the variance at index i.

$N(0,1)$ is a normally distributed one-dimensional random variable having expectation zero and standard deviation 1.

The strategy variable (i.e variance) is the mutated as follows

$$V_i' = V_i \cdot exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$

where

V_i' is the value of the mutated variance at index i.

V_i is the current value of the variance at index i.

τ' is commonly proportional to$(\sqrt{2n})^{-1}$, where n is the number of dimensions in the object variable.

τ is commonly proportional to $(\sqrt{2\sqrt{n}})^{-1}$, again n is the number of dimensions in the object variable.

$N(0,1)$ is a normally distributed one-dimensional random variable having expectation zero and standard deviation 1.

$N_i(0,1)$ is as above but the variable is sampled anew for each value of i.

Selection in EP consists of pair wise comparison over the union of parents and offspring (2μ individuals). Each individual is compared to q randomly chosen opponents. The number of wins for the individual in each of these tournaments is recorded. The top μ individuals with the highest number of wins go forward to form the next generation.

Traditional approaches to discrete optimisation [3] involves using a uniform random distribution to select one of a number of mutation operators. Discrete optimization with self-adaption builds on this approach by incorporating a probability mass function into the individual. The probability mass function determines the extent of the mutation. This probability mass function evolves by perturbing the parents probability using a Gaussian random vector with zero mean and an arbitrarily chosen variance. The perturbed probabilities are scaled using a scaling factor so that the sum of all probabilities is one.

To facilitate self-adaption each individual in the population consists of a genome and a number of strategy variables which evolve with the genome through successive generations. A strategy variable can influence the genome in any number of ways, for example, the rate of mutation to be applied to the genome could evolve as a strategy variable rather than being a fixed quantity as in the case of a Genetic Algorithm. Another example would be the use of a strategy variable to determine where in the genome string the mutation will take place.

The process of generating the offspring consists of first copying an individual (genome and strategy variable), then randomly perturbing the strategy variables and finally using the strategy variables to mutate the genome. The algorithm for self-adaptive discrete optimisation is given below.

1. Randomly create an initial population of n solutions.

2. Randomly create the initial strategy variables for each member of the population.

3. Create n offsprings by making a copy of the population and its associated strategy variables.

4. Mutate the strategy variables using a Gaussian random vector.

5. Mutate the offsprings using the mutated strategy variables.

6. Combine the original population and the mutated offsprings.

7. Determine the fitness of each member of the combined population.

8. Select the best n individuals from the combined population.

9. Go to step 3 until the required solution has been found or the maximum generations have been reached.

3.8 Evolutionary Strategies

Evolutionary Strategies (ES) emerged from the joint development of Bienert, Rechenberg and Schwefel in the 1960s at the Technical University of Berlin. Two principal forms of the strategy emerged, the $(\mu + \lambda) - ES$ strategy (elitist strategy) allows the best μ individuals from the union of parents (μ) and offspring (λ) to survive while the (μ, λ) strategy allows the best (μ) offspring's survive to the next generation. A significant part of ES is the capability for self-adaption, this is achieved by allowing the strategy variables evolve along with the object variables.

An individual in ES consists of an object variable vector (x) and optional strategy variables consisting of a number of standard deviations (σ) and

a number of rotation angles (α). Parents produce offspring by applying normally distributed mutations, these mutations control the step size, which in turn determines the mutability of the object variables.

Learning takes place at two levels, both in the optimisiation of the objective variables and the strategy variables. The strategy variables represent the standard deviation of a $(0, \sigma_i)$ Gaussian distribution with an expectancy value of 0. With this the parents should produce offsprings similar to themselves on average. Mutation can involve all three elements of the individual, the object variable vector, the standard deviations and the rotation angles. The mutation of the standard deviation strategy variables involves a multiplicative, logarithmic normally distributed process as follows:

The mutation of the standard deviations proceeds as follows:

$$\sigma_i^{'} = \sigma \cdot exp(\tau^{'} \cdot N(0,1) + \tau \cdot N_i(0,1))$$

where

$\sigma_i^{'}$ is the resulting mutated value of the standard deviation.

σ_i is the current value of the standard deviation.

τ is a factor suggested by Schwefel to be set $\propto (\sqrt{2\sqrt{n}})^{-1}$ where n is the number of dimensions in the object variable.

$\tau^{'}$ is suggested by Schwefel to be set to $\propto (\sqrt{2n})^{-1}$.

$N(0,1)$ is a normally distributed one-dimensional random variable having expectation zero and standard deviation 1.

$N_i(0,1)$ is as above but the variable is sampled anew for each value of i.

The mutation of the rotation angle proceeds as follows:

$$\alpha_j^{'} = \alpha_j + \beta \cdot N_j(0,1)$$

where $\alpha_j^{'}$ is the resulting mutated value of the rotation angle.
α_j is the current value of the rotation angle.
β is a constant recommended by Schwefel to be set at ≈ 0.0873.
$N_j(0,1)$ is a is a normally distributed one-dimensional random variable having expectation zero and standard deviation 1 which is sampled anew for each value of j.

The mutation of the object variable vector is as follows:

$$\vec{x}^{'} = \vec{x} + \vec{N}(\vec{0}, C(\vec{\sigma}^{'}, \vec{\alpha}^{'}))$$

where $\vec{x}^{'}$ is the resulting mutated object variable vector.
$\vec{x}$ is the current object variable vector.
$C(\vec{\sigma}^{'}, \vec{\alpha}^{'})$ is a covariance matrix of standard deviations $\vec{\sigma}^{'}$ and rotation angles $\vec{\alpha}^{'}$.
$\vec{N}(\vec{0}, C(\vec{\sigma}^{'}, \vec{\alpha}^{'})$ is a normally distributed random variable having expectation zero and standard deviation $C(\vec{\sigma}^{'}, \vec{\alpha}^{'})$.

For the case where rotation angles are not used the formula reduces to:

$$\sigma_i^{'} = \sigma \cdot exp(\tau^{'} \cdot N(0,1) + \tau \cdot N_i(0,1))$$

$$\vec{x}^{'} = \vec{x} + \sigma_i^{'} \cdot N_i(0,1)$$

A significant difference between ES and EP (see Section 3.7) is the use of recombination. A number of different crossover or recombination strategies are used in ES. Schwefel recommends discrete recombination on object variables and panmictic intermediate recombination on strategy parameters.

Discrete recombination (see Figure 3.7) involves selecting two parents at random from the population and creating an offspring, which consists of

object variables genes drawn with equal probability from both parents. More formally:

$$X_i' = X_{S,i} or X_{T,i}$$

where

X_i' is the value of the offsprings object variable at index i.

$X_{S,i}$ is the value of the first randomly selected parent at index i.

or in this context indicates that either parent can be selected with equal probability.

$X_{T,i}$ is the value of the second randomly selected parent at index i.

Panmictic (global) intermediate recombination (see Figure 3.8) involves selecting one parent at random from the population and creating an offspring through recombination with other randomly selected parents. These other parents are chosen randomly for each instance of the index i, again formally stated:

$$X_i' = 0.5 \times (X_{S,i} + X_{T_i,i})$$

where

X_i' is the value of the offsprings object variable at index i.

$X_{S,i}$ is the value of the first randomly selected parent at index i.

$X_{T_i,i}$ is the value of another randomly selected parent at index i. A different parent is randomly selected anew for each value of the index i.

There are two preferred methods of selection used in ES as indicated above. The $(\mu + \lambda) - ES$ strategy allows the best μ individuals from the

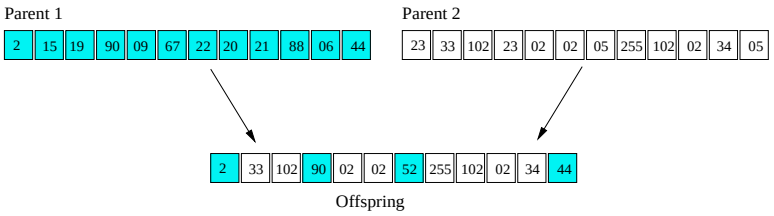

Figure 3.7: Discrete Recombination, the offspring is formed by codons from either parent with equal probability of selection.

union of parents (μ) and offspring (λ) to survive while the (μ, λ) strategy allows the best (μ) offspring's survive to the next generation. The (μ, λ) strategy which limits individuals to one generation is generally preferred [1].

The ratio μ/λ determines the convergence property of the evolutionary strategy. Decreasing μ leads to path orientated search (predictive or exploratory methods providing fast local convergence) while increasing μ leads to a volume orientated search (assumes the whole feasible region must be scanned which increasing chances of finding a global optimum) [1] pages 46-47. The logarithmic normal distribution for the variations of standard deviations insures that smaller modifications must occur more often than larger ones.

3.8.1 Relationship between EP and ES

ES like EP has primarily been associated with optimisation of continuous data. Optimisation of discrete data can be achieved by using a uniform random distribution to select one of a number of mutation operators. As with EP, an element of self-adaption can be introduced by incorporating a probability mass function into the individual. The probability mass function which determines the extent of the mutation evolves by perturbing the parents probability using a Gaussian random vector with zero mean and an

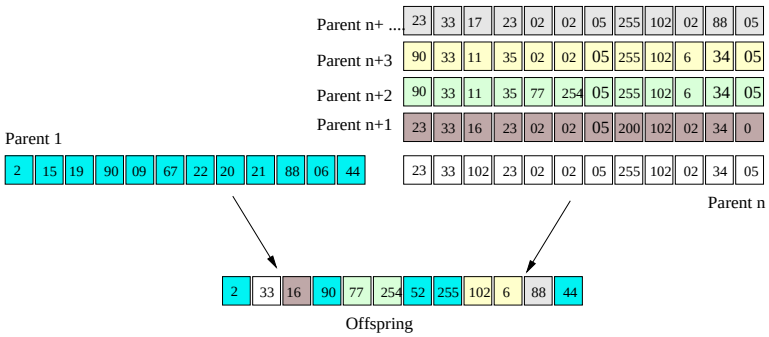

Figure 3.8: Intermediate Recombination, the offspring is formed by codons from a selected parent and n other randomly (equal probability of selection) selected parents.

arbitrarily chosen variance. The perturbed probabilities are scaled using a scaling factor so that the sum of all probabilities are one.

3.8.2 Self Adaptive Discrete Optimisation

Each individual in the population consists of a genome and a single strategy variable. This strategy variable determines the rate of mutation to be applied to the genome. The strategy variable which evolves along with the genome, consists of an integer value that corresponds to the number of codons altered during mutation, The process of generating the offspring involves copying an individual (genome and strategy variable), then randomly perturbing the strategy variable and finally using the strategy variable to determine the rate of mutation to be applied to the genome. The algorithm for self-adaptive discrete optimisation which is the form of EP used in these experiments is given below.

1. Randomly create an initial population of n solutions.

2. Randomly create an initial strategy variable for each member of the

population.

3. Using selected recombination method create $2n$ offsprings.
4. Mutate the strategy variable using a Gaussian random vector.
5. Mutate the offsprings using the mutated strategy variables.
6. Determine the fitness of each member of the combined population.
7. Select the best n offspring to replace the parent population.
8. Go to step 3 until the required solution has been found or the maximum generations have been reached.

3.9 Summary

In this chapter we have introduced each of the selected metaheuristics. Genetic Algorithms, Genetic Programming, Evolutionary Programming and Evolutionary Strategies are population-based methods that evolve populations of solutions using a combination of selection, mutation and recombination, while Simulated Annealing and Hill Climbing are local search techniques that explore the search space by modifying a single individual. Random search is, in essence, a sampling of the search space, providing a useful indication of the density of solutions and accordingly the relative difficulty of problems. In our examination of these techniques we have presented the basic algorithm involved and discussed the operators and selection strategies that charachterise each method.

Chapter 4

Problem Domains

4.1 Introduction

This chapter introduces each of the selected problems. A description of each problem is provided as well as the grammar used by GE in its attempts to solve the problem is presented in Backus Naur Form. The Santa Fe trail problem consists of an ant searching for food on the Santa Fe Trail, Symbolic Integration and Symbolic Regression attempt to find functions to fit data sets and Blocks is an attempt to stack an ordered set of nine blocks.

4.2 Santa Fe Trail

The Santa Fe Trail is a benchmark problem in Genetic Programming (GP) [14]. The objective is to devise a program which can navigate along a 32 X 32 toroidal grid picking up pieces of food positioned on the grid. The ant can move left, right, or straight ahead, and can sense food directly in front of it. This ability to sense food is used to select alternate moves to guide the search. For the purpose of this evaluation the ant is allowed 615 time steps, with the

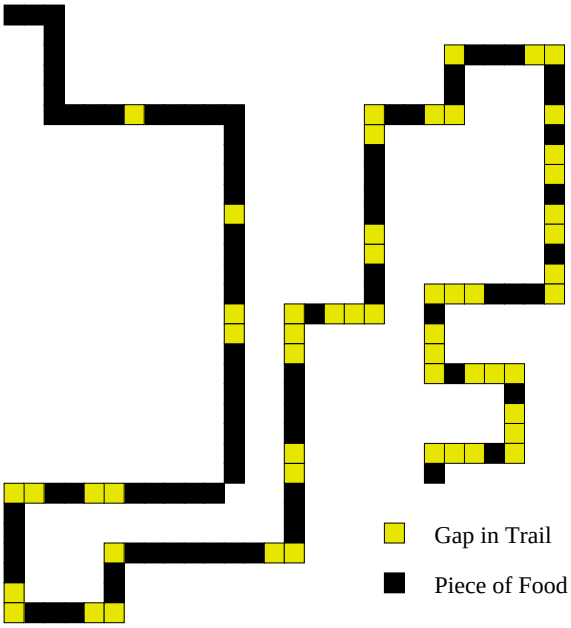

Figure 4.1: Sante Fe Trail.

exception of sensing food in the square ahead each action consumes one time step. The fitness is the number of pieces of food (of a possible 88) collected within the 615 time steps. Figure 4.1 shows the trail used in these trials.

Previous research [18] has shown that there is a high density of distinct solutions for this particular problem, with the neighbourhoods of these solutions composed of low fitness programs and a large number of sub-optimal peaks. The BNF grammar for this problem is shown below.

```
<code>          ::== <line> | <code><line>
<line>          ::== <condition> |  <op>
<condition>     ::== if(food_ahead()){ <line> else <line>}
<op>            ::== left() | right() | move()
```

4.3 Symbolic Integration

Symbolic Integration involves finding a function that is the integral of the given curve. The system uses a set of input and output pairs, where the objective is to find the function that maps the inputs to the outputs. For these trials the particular function used was:

$$f(x) = Cos(x) + 2x + 1$$

with the input values in the range $[0...2\pi]$. The target curve was

$$f(x) = Sin(x) + x^2 + x$$

The fitness for this problem is given by the sum of the absolute value of the difference, taken over 20 fitness cases, between the individual genetically produced function $f_j(x_i)$ at the domain point x_i and the value of the numerical integral $I(x_i)$. The grammar used is shown below.

```
<expr>          :: == <var> | <expr><op><expr> | <pre-op>(<expr>)
                      | (<expr>)
<var>           :: == X
<op>            :: == + | - | / | *
<pre-op>        :: == sin | cos | tan | log
```

4.4 Symbolic Regression

The objective of the Symbolic Regression problem is to find a function of one independent variable and one dependent variable in symbolic form that fits a sample of $20(x_i, y_i)$ data points in the range [-1,+1]. The target function

is that used by Koza [14].

$$x^4 + x^3 + x^2 + x$$

The grammar used is identical to that shown for Symbolic Integration.

4.5 Block Stacking

The objective of the block-stacking problem is to produce a specified ordered set of nine blocks stacked as a tower. The starting position involves an initial stack of blocks with the remainder on the table. Figure 4.2 shows a typical starting position.

Three lists are used to specify the problem, the GOAL-LIST specifies the desired final order in which the blocks are to be stacked in the target tower (i.e. "UNIVERSAL"). The STACK-LIST is the ordered set of blocks that are currently in the target tower (where the order is important). The TABLE-LIST is the set of blocks that are currently not in the target tower (where the order is not important). The initial configuration consists of certain blocks in the STACK-LIST and the remaining blocks in the TABLE-LIST. The desired final configuration consists of all the blocks being in the STACK-LIST in the order specified by the GOAL-LIST and no blocks being in the TABLE-LIST.

Three sensors are used to track the system in the formulation of the problem, TB (Top Block) refers to the highest correct block on the stack, NN (Next Needed) is the next block required on the stack as dictated by the specified ordered set of blocks and CS (Current Stack) refers to the top block of the stack. Each of these sensors can assume one of the nine block labels or NIL. These are illustrated in Figure 4.5.

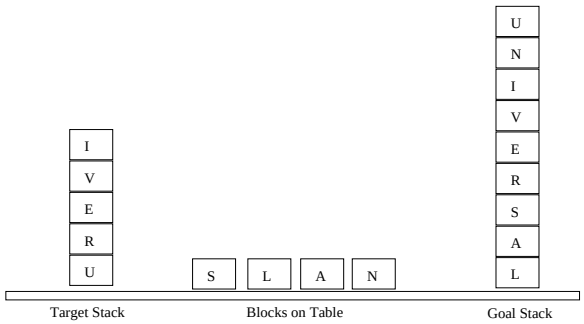

Figure 4.2: Typical Starting Position for the Blocks Problem.

Five functions are available to solve the problem, including NOT (!) and EQUAL (EQ) the usual Boolean negation and equality operators, MS (Move Stack, see Figure 4.3) moves a single block from the table to the stack. It takes a single argument X as determined by one of the three sensors. If X is on the table then X is moved to the top of the stack and MS returns true. If X is already in the stack, if the table is empty or if X itself is NIL MS does nothing and returns NIL. MT (Move Table, see Figure 4.4) moves a block from the top of the stack to the table. It takes a single argument X as determined by one of the three sensors. If X is in the stack MT moves the top of the stack to the table and returns true. If X is already on the table, if the table is empty or if X itself is NIL MT does nothing and returns NIL. Finally DU (Do Until) takes two arguments, a body of code that it iteratively executes until the second argument (a predicate) becomes true.

In order to guard against predicates that never evaluate to true an iteration limit is used. This takes the form of 25 iterations for a single DU loop or 100 iterations for all DU operations associated with a particular individual. These limits are those used by Koza [14] in his evaluation of the problem using GP. If the DU loop is terminated by the predicate it returns true, termination by the iteration threshold forces the DU loop to return false. The

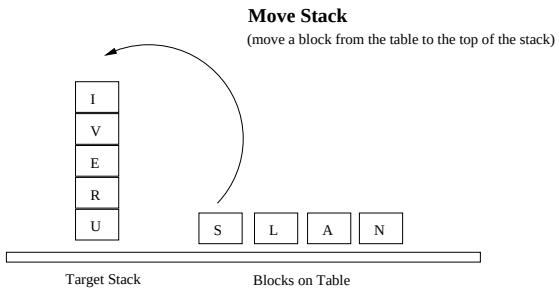

Figure 4.3: Move Stack operation.

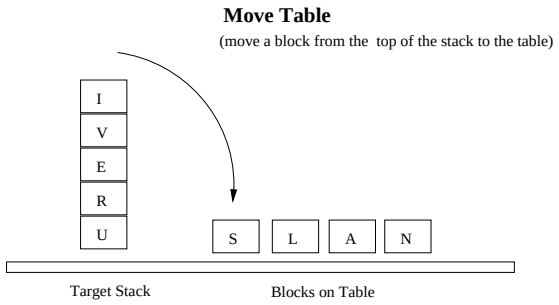

Figure 4.4: Move Table operation.

Grammar associated with this problem is shown below.

Ten fitness cases are used in this problem. Three of these (the case of the initially correct stack, the empty stack and the empty table) are pre-selected. The other seven are randomly generated. The scores are allocated as follows. Each empty stack is allocated one point, each stack with at least one letter in the correct order is allocated two points, and each fully correct stack is allocated three points. Based on this allocation of points a fully fit individual will score thirty points.

```
<code>          ::== <line> | <code> <line> | <op> <code>  <code>
                   | <pre-op> <code> | <op> <var> <var>
<line>          ::== <control> |  <expr>
<expr>          ::== MS ( <var> ) | MT ( <var> )s
<control>       ::== DO  <code> UNTIL <code> ODU
```

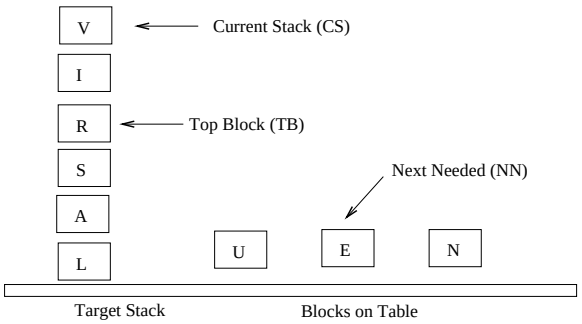

Figure 4.5: Move Table operation.

```
<op>            ::== EQ
<pre-op>        ::== !
<var>           ::== TB | NN | CS
```

4.6 Summary

In this chapter we have introduced each of the four selected problems, the Santa Fe trail problem, Symbolic Integration, Symbolic regression and Blocks. The problems which are all well-known bench mark problems in the field of Genetic Programming were presented with a brief summary of some of the objectives, characteristics, method for scoring and a summary in Backus Naur Form of the grammar used by GE to attempt to solve them.

Chapter 5

Random Search

5.1 Introduction

This chapter focuses on the performance and characteristics of Random Search. The experimental conditions are first presented before detailing the results with some discussion on aspects of the solutions found. The degree to which wrapping occurs in the successful trials is examined and the impact on the success rate of its removal is also looked at.

One of the key objectives of this chapter is to provide some insight into the relative difficulty of the selected problems. This is based on the assumption that Random Search as a method of sampling the search space provides some indication of the solution density associated with a particular problem.

5.2 Search Strategy Options

For Random Search, individuals of random length and random value are created and evaluated. The size of the sample is fixed at 25000, which is the number of evaluations of the objective function that have been allowed per

trial to each of the search methods evaluated in this research.

5.3 Experimental Conditions

Table 5.1 shows the parameters used to configure Random Search for the trials. For Symbolic Integration, Santa Fe Trail and Blocks a maximum genome length of 100 was used, while Symbolic Regression used a figure of 200. The selected values for maximum length were influenced by a consideration of the grammar used and the likely nature of the target expression for each of the problems.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length	100	100	100	200
Initial Genome Length Variation Range	10%	10%	10%	10%
Wrapping	on	on	on	on

Table 5.1: Parameters used to configure Random Search.

5.4 Results

The results of the trials are shown in Table 5.2. Symbolic Integration proves to be the easiest problem to solve with a success rate of 40%. The Santa Fe Trail also scores a high level of success with a figure of 30% while Blocks is

solved in 24% of the attempts. Symbolic Regression was not solved in any of the trials.

Problem	Successful Runs
Symbolic Integration	40%
Santa Fe Trail	30%
Blocks	24%
Symbolic Regression	0%

Table 5.2: Results from Trials of Random Search.

A significant aspect of these results is the insight they provide into the relative difficulty of the problems. The success rates for Symbolic Integration, Santa Fe and Blocks would suggest a high density of solutions within the search space for these problems.

A more detailed examination of Random Search in the case of Symbolic Integration and the Santa Fe Trail indicates that there is no revisiting of previously encountered individuals, that is each of the 25000 samples are unique. This lack of duplication is maintained even when one reduces the maximum permitted value of the codon to the minimum possible, that is the number of productions in the grammar (i.e. removal of Genetic Code Degeneracy see Section 2.5). For example fixing the maximum codon value to 2 in the case of Santa Fe and 3 in the case of Symbolic Integration.

Another emerging feature from these results is the correlation between average number of expressed codons in a solution and success rates. Symbolic Integration proves to be the easiest of the problems to solve, requiring on average just 14 codons (see Table 5.3) while Santa Fe and Blocks require on average 46 and 69 codons respectively.

5.5 Characteristics of Solutions found by Random Search

Table 5.3 shows some of the characteristics of the solutions found by Random Search. Symbolic Integration requires on average only 14 codons, while Santa Fe and Blocks require 46 and 69 respectively. Wrapping occurs in 43% of Santa Fe solutions and 41% of Blocks solutions while none of the Symbolic Integration solutions use wrapping. Wrapping on the Blocks problem often results in multiple wrap events on the same genome, averaging two wrap events per succesful solution. Santa Fe averages just one wrap event for successfull solutions.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	57	57	58	n/a
Avg Number of expressed Codons in Solution	14	46	69	n/a
Percentage of Solutions featuring Wrapping	0%	43%	41%	n/a

Table 5.3: Analysis of Characteristics from Solutions found by Random Search.

5.6 Impact of wrapping

Wrapping appears prominently in the Santa Fe and Blocks problems with 43% of the Santa Fe solutions and 41% of Blocks solutions featuring wrapping.

Symbolic Integration doesn't use wrapping in any of the solutions found by Random Search. An examination of the average number of expressed codons used in solutions to the Symbolic Integration problem shows it to be quite short, at 14, which contrasts with the figure of 57 for the average number of codons used in a successful genome. This ratio of actual codons provided to actual codons required is greatest for Symbolic Integration. Comparing this same ratio for Santa Fe and Blocks reveals that although both problems use wrapping at similar levels (41% - 43%) the ratio for Blocks is actually less than one indicating a much higher number of wrap events. An analysis of the Blocks solutions supports this, revealing up to four wrap events within a single solution on occasion.

Column 2 of Table 5.4 shows the impact on success rates of removing wrapping. Symbolic Integration which showed no dependency on wrapping remains unchanged while Santa Fe and Blocks drop by 15% and 7% respectively.

Problem	Successful Runs	Change in Success Rate	Avg Number of Codons	Avg Number Codons Used
Symbolic Int.	41%	+1%	53	15
Santa Fe Trail	15%	-15%	63	37
Blocks	17%	-7%	65	39
Symbolic Reg.	0%	n/a	n/a	n/a

Table 5.4: Results from 1000 trials of Random Search with Wrapping disabled on the problem set.

5.7 Summary

In this chapter we have looked at the efforts of Random Search in solving the problem set. Symbolic Integration, Santa Fe Trail and Blocks were solved with surprisingly high success rates, while all trials on Symbolic Regression and Spirals were unsuccessful. Wrapping featured prominently in the results for Santa Fe and Blocks, removing wrapping sees a drop in the success rate for these two problems. If we regard Random Search as a means of providing insight into the density of solutions in the search space then we can conclude that Symbolic Integration requiring an average of 14 codons has the highest density of solutions making it the easiest of the selected problems. Santa Fe and Blocks also show high solution densities requiring an average of 46 and 69 codons respectively. The results suggest that Symbolic Regression is a much more difficult problem, remaining unsolved by random search.

Chapter 6

Hill Climbing

6.1 Introduction

In this chapter we focus on the performance of the Hill Climbing search algorithm. Hill Climbing is a non-population based method starting from a single initial solution. This feature coupled with the fact that we are using fine-grained operators makes the starting point in the search space very important. We look initially at the strategy options and examine the parameters used to configure the Hill Climbing algorithm. The results section shows the performance of the algorithm and examines the consequences of different strategy choices for the algorithm in terms of efficiency and success. Initial genome length, zero improvement acceptance, wrapping and the contribution of different operator probabilities are all examined.

6.2 Search Strategy Options

As in the case of Random Search we fix each trial at 25000 evaluations of the objective function. The operators chosen for Hill Climbing have been

designed to allow fine-grained exploration of the search space. Force-up, force-down, shrink and grow (see Section 3.6 for a more detailed explanation) operate on a single codon at each iteration, allowing close control of the evolving solution. The four operators are assigned equal probability of occurrence.

6.3 Experimental Conditions

Table 6.1 shows the parameters used to configure Hill Climbing. Initial genome length is set at 100 for each of Symbolic Integration, Santa Fe Trail and Blocks, while the more difficult problem of Symbolic Regression is permitted an initial genome length of 200. *Zero improvement acceptance*, which is the strategy of accepting solutions whose score is the same as the current score, it is set to off (i.e. do not accept) for the initial trials.

6.4 Results

The summary results provided in Table 6.2 show Blocks and Santa Fe Trail as the problems with the highest success rate at 17% and 15% respectively. Symbolic Integration has limited success at only 8% while Symbolic Regression cannot be solved by the algorithm. These figures are significantly lower than the scores recorded by Random Search in the previous chapter, where Symbolic Integration scored 40%, Santa Fe scored 30% and Blocks scored 24%. In an effort to try and understand this difference and improve the performance of Hill Climbing a number of strategy options are analysed in the following sections.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length	100	100	100	200
Initial Genome Length Variation Range	10%	10%	10%	10%
Zero Improvement Accept	off	off	off	off
Probability of Selecting Shrink Operator	.25	.25	.25	.25
Probability of Selecting Grow Operator	.25	.25	.25	.25
Probability of Selecting Force-up Operator	.25	.25	.25	.25
Probability of Selecting Force-down Operator	.25	.25	.25	.25

Table 6.1: Parameters used to configure the Hill Climbing Search Algorithm.

6.4.1 Impact of Zero Improvement acceptance

Zero improvement acceptance, the strategy of accepting solutions whose score is the same as the score of the current best performing genome could have provided a means of more extensive exploration of the search space, particularly given the potential for neutral mutation in GE, however permitting acceptance of zero-improvement solutions has no statistically significant impact on success as shown in row three of Table 6.3.

Problem	Successful Runs
Symbolic Integration	8%
Santa Fe Trail	15%
Blocks	17%
Symbolic Regression	0%
Spirals	0%

Table 6.2: Results from the Hill Climbing Trials.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Accept Zero Improvement Solutions	8%	17%	11%	n/a
Reject Zero Improvement Solutions	8%	15%	17%	n/a
Difference	0%	+2%	-6%	n/a

Table 6.3: Impact of Zero Improvement Acceptance on the Hill Climbing Search Algorithm.

6.4.2 Impact of Initial Genome Length

One of the motivating factors behind the selection of the force-up, force-down, shrink and grow operators was the provision of a mechanism that would allow Hill Climbing create solutions through fine controlled adjustment of the codons within a genome. In this respect it is useful to look at the impact of initial genome length on performance.

One of the assumptions with Hill Climbing was that a smaller genome length would suit the algorithm allowing it grow the solution using the fine-

grained operators. This would for to be a more efficient use of the given allocation of objective function evaluations rather than starting with a long genome that failed to map and then attempting to shape it, as the initial expression did.

Table 6.4 shows the impact of initial genome length on the Symbolic Integration problem when Hill Climbing is employed. Initial genome lengths of 5, 8, 10, 12, 14, 16 and 18 were chosen. Analysis of the results indicate that success rate is independent of initial genome size for this problem.

Initial Genome Length	Successful Runs
5	8.4%
8	8.1%
10	8.2%
12	9.4%
14	6.9%
16	6.0%
18	7.6%

Table 6.4: Effect of Variation of Initial Genome Length on Success Rate for Hill Climbing on the Symbolic Integration Problem.

An analysis of these results show that the insignificance of initial genome length is a consequence of the large number of iterations required for Hill Climbing to find a solution. Typically Hill Climbing solutions will have undergone significant variations in size by the time a solution is found. This is evidenced by the plot of genome length and fitness for a successful trial shown in Figure 6.1, where the genome is shown to expand and contract as the algorithm moves through the search space. Even with a short starting genome there is always ample time for the solution to shrink and grow

through successive iterations. Another feature of this profile is a noticeable decrease in the length of genomes in later iterations.

6.4.3 Influence of Operators on Success

The performance of Hill Climbing is poor relative to that of Random Search in the last chapter. In an effort to understand some of the reasons for this an analysis of the contribution of the operators was undertaken.

On approach to looking at the contribution of operators is to examine the efficiency of the algorithm as it moves through the search space. Does it, for example, efficiently explore the space? Or, does it spend time re-visiting the same solutions? A measure of this is the degree of duplication of solutions in any given trial.[1]Table 6.5 shows a figure of 8.4% duplication in the initial trials on the Symbolic Integration problem. An analysis of specific search trajectories show that the main cause of this duplication is successive occurrences of the shrink and grow operators where a genome first grows in one direction before shrinking back on itself. This is also graphically illustrated in Figure 6.1 which shows the oscillating nature of the search.

Reducing the probability of the shrink operator should have the effect of reducing duplication with a possible increase in success rates. Table 6.5 illustrates this in the case of Symbolic Integration where the success rate climbs from 8% to 14% when the probability of the shrink operator is reduced to .05 (5%). We also see the number of duplicate solutions fall from 8.4% to 1.6% under the influence of this change. Tables 6.6 and 6.7 show a similar situation for the Santa Fe and Blocks problems. The Symbolic Regression

[1]The issue of re-visiting previously evaluated solutions is specifically managed in Tabu Search by a mechanism called the Tabu List. The Tabu List is a list of recently visited solutions. The search algorithm is prevented from visiting solutions on the Tabu List. [7]

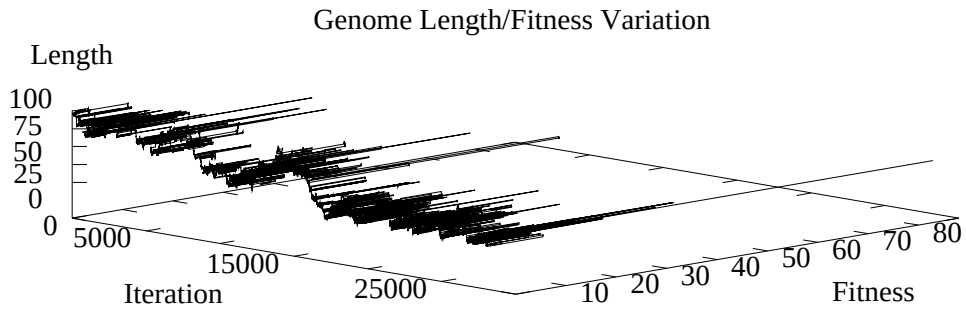

Figure 6.1: Variation in Genome Length for a Successful Hill Climbing Trial on the Santa Fe Trail problem.

problem remained unsolved when operator probabilities were modified so a results summary for this problem has not been included.

Interestingly while this reduction in the probability of the shrink operator sees similar reduction in duplication for the Santa Fe and Blocks problem we do not see any significant change in the success rate. To further underline the fact that reduced duplication in itself does not guarantee higher success rates we can look at the consequences of introducing a coarser operator into the Hill Climbing algorithm. The introduction of *multiple mutation*, which permits the changing of up to 10% of a genome's codons in a single iteration, sees the amount of duplicate solutions rise to 13% while the success rate also rises to 18% for the Symbolic Integration problem. One has to be cautious here however as the introduction of such a disruptive operator moves the algorithm closer to random search rather than the controlled exploration of the search space envisaged for Hill Climbing.

Operators	Successful Runs	Percentage Duplication
shrink and grow at .25	8%	8.4%
shrink and grow at .5	14% (+6%)	1.6% (-6.8%)
multiple mutation 10%	18% (+10%)	13% (+4.6%)

Table 6.5: Effect of variation of operators on success rate for Hill Climbing on the Symbolic Integration problem.

Operators	Successful Runs	Percentage Duplication
shrink and grow at .25	15%	8%
shrink and grow at .05	11% (-4%)	2% (-6%)
multiple mutation 10%	18% (+3%)	13% (+5%)

Table 6.6: Effect of variation of operators on success rate for Hill Climbing on the Santa Fe Trail problem.

6.5 Characteristics of Solutions found by Hill Climbing

An examination of the data presented in Table 6.8 shows a high percentage of solutions for Santa Fe (80%) and Blocks (70%) that use wrapping. Despite starting with an initial genome length of 100 (+/- 10%) the Hill Climbing algorithm on these problems consistently moves towards genomes with smaller numbers of codons. In the case of Hill Climbing this move is facilitated by the *shrink* operator. Even when the contribution of this operator is reduced from 25% to 5% on the Santa Fe Trail problem the high incidence of wrap-

Operator Probabilities	Successful Runs	Percentage Duplication
shrink and grow at .25	17%	9.3%
shrink and grow at .05	16% (-1%)	3.5% (-5.8%)
multiple mutation 10%	10% (-7%)	14% (+4.7%)

Table 6.7: Effect of reduced probability for shrink and grow operators on success rate for Hill Climbing on the Blocks problem.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	52	42	50	n/a
Avg Number of expressed Codons in Solution	15	47	77	n/a
Percentage of Solutions featuring Wrapping	0%	80%	70%	n/a

Table 6.8: Analysis of Characteristics from Solutions found by Hill Climbing.

ping is maintained at 46% of solutions with the average number of codons rising to 58.

If the ability to migrate to short genomes is eliminated by removing the *shrink* operator altogether the success rate of Hill Climbing on the Santa Fe Trail problem fall significantly from 15% to 7%. In the case of Symbolic Integration wrapping is not involved in any of the solutions, with the average number of expressed codons (15) less than half the size of the average number of codons in a solution (52). Removing the *shrink* operator in this instance sees no change in the success rate. This should be expected given the absence of wrapping found in the original trials.

A curious aspect of these results is the tendency toward shorter genome lengths when GE could just as easily ignore the unexpressed codons to the right. An examination of the search trajectory however reveals situations in which we typically see large numbers of iterations between changes in the score, during this period the genome shrinks and grows repeatedly under the influence of the shrink and grow operators. It is during this period when the genome is shortest that we are more likely to change a codon that results in an improved score.

In the case of Blocks, which have the highest percentage of wrapping, the number of expressed codons is far greater than the average number of codons found in a solution. An analysis of the results shows that solutions often use multiple wrap events. Some of the underlying causes of this will be explored in later chapters.

6.5.1 Analysis of search trajectory

A clear indication of the difficulty faced by Hill Climbing on these problems is provided by an analysis of the search trajectories of successful Hill Climbing trials. A representative plot is shown in Figure 6.2.

This plot of the search trajectory shows the Hill Climbing algorithm negotiating its way through low scoring solutions before making a quantum leap to the optimum solutions.

6.6 Problem Difficulty

The results for Hill Climbing also challenge some of the assumptions we made about problem difficulty in the previous chapter, where the results from Random Search suggested that Symbolic Integration was the easiest of

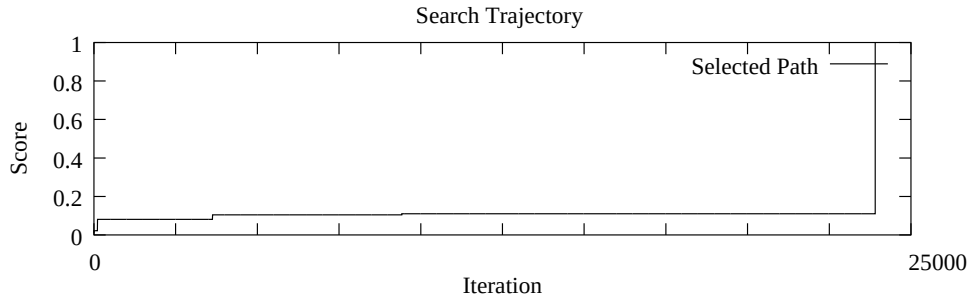

Figure 6.2: Search Trajectory for a successful Hill Climbing trial on the Santa Fe Trail Problem

the problems to solve. Hill Climbing scored 8% on this problems against a score of 40% for Random Search, indicating that the difficulty of a problem is relative to the search method being used. Again, this is supported by the results from the Santa Fe trail and Blocks problems where Hill Climbing scores 15% and 17% respectively while Random Search clearly finds Santa Fe easier (scoring 30%) than Blocks (scoring 24%).

6.7 Summary

In this chapter we have looked at the performance of Hill Climbing a local search Metaheuristic. The impact of accepting or rejecting *zero improvement* solutions was examined and found to have no significant effect. We also looked at the influence of the length of the initial Genome on success rates and examined the contribution of the selected operators in terms of efficiency and effectiveness. We looked at the tendency toward short genomes lengths and the use of wrapping to find solutions. This was particularly evident in the case of Santa Fe Trail and Blocks.

Overall the performance of Hill Climbing has been poor relative to the efforts of Random Search in the previous chapter. Even when it does succeed

we see a quantum leap from very low fitness to maximum fitness rather than a gradual ascent through intermediate levels of fitness. While many of the attempts to modify the strategy options associated with Hill Climbing saw no significant improvement in the success rate, the introduction of a more disruptive mutation operator did see a significant increase in the score. Finally, we have re-assesed some of our assumptions about problem difficulty based on the comparative results of Hill Climbing and Random Search.

Chapter 7

Simulated Annealing

7.1 Introduction

The Simulated Annealing search algorithm is examined in this chapter, looking initially at the strategy options and configuration choices. The results section shows the performance of the algorithm in preliminary trials with a range of annealing schedules before presenting the results of the main trials.

7.2 Search Strategy Options

To be consistent with efforts of Random Search and Hill Climbing we again limit each trial to 25000 evaluations of the objective function. The operators used for Simulated Annealing are those that were used with Hill Climbing, however we have reduced the probability of occurrence of the shrink and grow operators based on our observations regarding duplication of solutions as discussed in Section 6.4.3.

The approach in general is similar to Hill Climbing, as the operators have been selected to allow fine-grained exploration of the search space while

exploiting the capability of Simulated Annealing to avoid the pitfalls of local optima. The principle strategy choice for Simulated Annealing will be the selection of a suitable annealing schedule (see Section 3.5), this is covered in detail in Section 7.4.

7.3 Experimental Conditions

Table 7.1 shows the parameters used to configure Simulated Annealing. Symbolic Integration, Santa Fe and Blocks use initial Genome lengths of 100 while Symbolic Regression uses an initial length of 200. A temperature change rate of 0.95 is used to reduce the temperature on each iteration. A minimum temperature of *0* is used to halt the annealing schedule. For a more complete explanation Section 3.5 provides a step-by-step guide to the algorithm.

7.4 Establishing an Annealing Schedule

The establishment of an appropriate annealing schedule for each of the problems centres on selecting the desired level of probability for acceptance of a dis-improving solution. At a temperature t, the probability P for a change in energy of magnitude δE is given by

$$P(\delta E) = e^{\frac{-\delta E}{kt}}$$

Figure 7.1 shows the change in the probability of accepting a dis-improving solution for a range of score deltas. Three different temperatures have been shown. If it was possible to predict the type of score deltas we were likely to see in the course of a trial then we could usefully employ this information to select a suitable starting temperature. We know from the analysis of the

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length	100	100	100	200
Variation Range	10%	10%	10%	10%
Zero Improvement Accept	off	off	off	off
Probability of Selecting Shrink Operator	.05	.05	.05	.05
Probability of Selecting Grow Operator	.05	.05	.05	.05
Probability of Selecting Force-up Operator	.45	.45	.45	.45
Probability of Selecting Force-down Operator	.45	.45	.45	.45
Temperature Change Rate	0.95	0.95	0.95	0.95
Minimum Temperature	0	0	0	0

Table 7.1: Parameters used for the Simulated Annealing Search Algorithm.

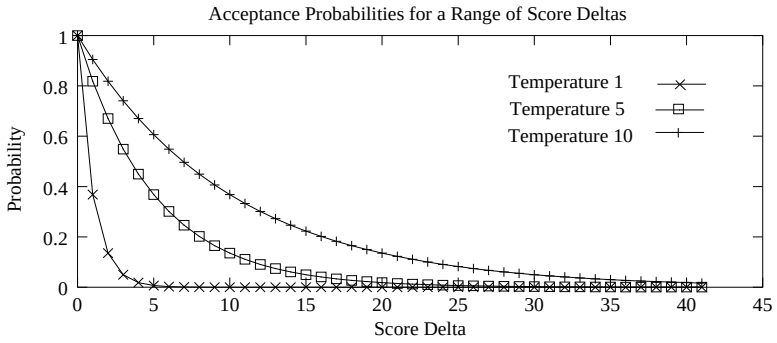

Figure 7.1: Probability of accepting a dis-improving Solution for a Range of Score Deltas at three different Temperatures on the Santafe Trail Problem.

Hill Climbing results that solutions often emerge from areas of the solution landscape where scores are low score. This might suggest that it is reasonable to accept large dis-improving deltas late in a trial. However this policy would also have the consequence of having us accept even larger score deltas early in a trial, this could force the search to remain trapped in sub optimal regions of the search space.

This is better illustrated in Figure 7.2 which again shows the change in the probability of accepting a dis-improving solution but this time it has been plotted against increasing temperature. Three plots for score deltas of 1, 5 and 10 have been included. Again we see some of the trade offs in this graph a high starting temperature exposes the search to higher probabilities of moving into low scoring regions of the search space while starting with a very low temperatures might leave the search unable to effectively explore the search space because of a overly captious approach.

In order to get some indication of suitable annealing schedules for the problems a limited number of runs were first performed for a range of starting temperatures. The number of iterations at each temperature accordingly varied in order to maintain our limit of 25000 evaluations of the objective

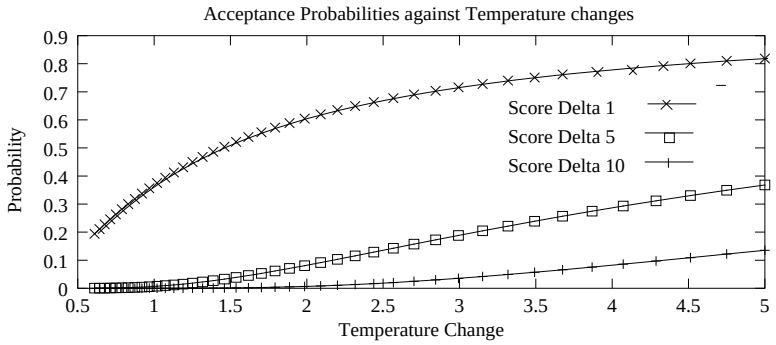

Figure 7.2: Probability of accepting a dis-improving Solution for a Range of Temperatures at three different Score Deltas on the Santafe Trail Problem.

function in a single trial. The results of these preliminary runs plus the results of the extended runs are presented in the next section.

7.5 Results

Tables 7.2, 7.3 and 7.4 show the results from the preliminary trials used to determine the appropriate annealing schedule for Symbolic Integration, Santa Fe and Blocks respectively. Similar runs on the Symbolic Regression problem failed to find any correct solutions.

These preliminary trials consisted of 500 runs for each start temperature. The table shows the starting temperature and the number of successes achieved.

For the case of Symbolic Integration the success rate appears to peak at 11% for starting temperatures of 1 with the success rate trailing off either side of this. On the Santa Fe problem we see a plateau of similar scores for temperatures of 1 or less with a significant fall off in success rates for higher starting temperatures. The schedules for Blocks follows a similar pattern to that of Santa Fe showing higher success rates at starting temperatures of

one or less. The Symbolic Regression problem was not solved in any of the preliminary trials.

Starting Temperature	Successful Runs
0.3	10%
0.1	6%
1	11%
5	5%
10	2%

Table 7.2: Results from 500 trials of Simulated Annealing on the Symbolic Integration Problem for a Range of Starting Temperatures.

Starting Temperature	Successful Runs
0.3	17%
0.1	20%
1	19%
5	4%
10	5%

Table 7.3: Results from 500 trials of Simulated Annealing on the Santa Fe Trail Problem for a Range of Starting Temperatures.

Table 7.5 provides a summary of the final results for Simulated Annealing. 1000 runs were performed using most successful starting temperature of 1 from the preliminary trials. Symbolic Regression was not solved in any of the attempts, however Symbolic Integration and Blocks have similar success rates scoring 11% and 13% respectively, while Santa Fe scores 21%. While the algorithm is more successful on the Symbolic Integration and Santa Fe

Starting Temperature	Successful Runs
0.1	11%
0.3	13%
1	12%
5	6%
10	9%

Table 7.4: Results from 500 trials of Simulated Annealing on the Blocks Problem for a Range of Starting Temperatures.

problems than Hill Climbing, the overall results are still poor relative to those of Random Search.

Problem	Successful Runs
Symbolic Integration	11%
Santa Fe Trail	21%
Blocks	13%
Symbolic Regression	0%
Spirals	0%

Table 7.5: Results from the Simulated Annealing Trials.

7.6 Characteristics of Solutions found by Simulated Annealing

Table 7.6 shows details of the solutions found by Simulated Annealing, wrapping features prominently in Santa Fe and Blocks. The average solution length for Santa Fe is 58 codons of which 48 are expressed. Symbolic Inte-

gration has solutions with an average genome length of 76 of which only 19 are expressed. An aspect of the results from the Blocks problem is that the number of expressed codons was 91, which is considerably greater than the average number of codons in a solution at 54. This aspect of the Block's results is caused by multiple wrap events where the codons are repeatedly re-used until a successful solution is found.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	76	58	54	n/a
Avg Number of expressed Codons in Solution	19	48	91	n/a
Avg Number of Solutions featuring Wrapping	0%	50%	66%	n/a

Table 7.6: Analysis of Characteristics from Solutions found by Simulated Annealing.

7.6.1 Analysis of search trajectory

A typical search trajectory for a Simulated Annealing trial is shown in 7.3. In the early stages the algorithm explores areas of both low and high fitness but as the trials progresses we see a steady climb to areas of high fitness as the decreasing temperature reduces the probability of selecting dis-improving solutions.

7.7 Summary

In this chapter we have looked at Simulated Annealing, a metaheuristic, which models itself on the way crystals form in solids during the cooling process. The principle strategy choice for the method was the selection of a suitable annealing schedule, to assist this selection a number of preliminary trials were performed using a range of starting temperatures. The final results showed unsuccessful attempts on two of the problem, Symbolic Regression and Spirals and low success rates on the other three.

The additional capability of Simulated Annealing to avoid sub-optima would appear to have added little or no improvement on the scores achieved by Hill Climbing, leaving it a considerable way behind Random Search.

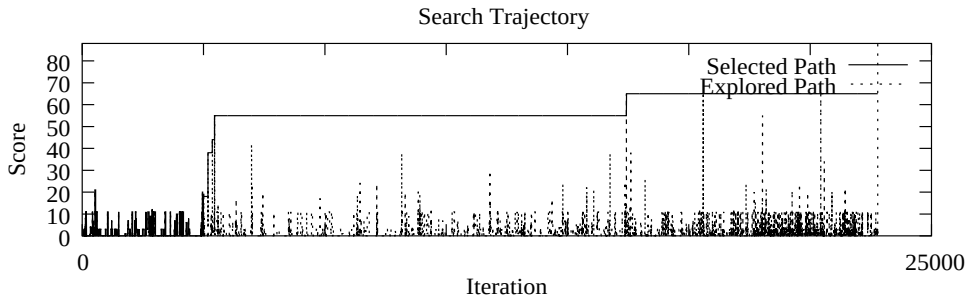

Figure 7.3: Search Trajectory for a Successful Simulated Annealing Trial

Chapter 8

Genetic Algorithms

8.1 Introduction

In this chapter we look at the performance of Variable-length Genetic Algorithms, the search strategy which has been used in all previously published work on GE. We begin with some discussion on the search parameters selected for Genetic Algorithms and then present the results of the trials with some analysis of the contribution made by the chosen operators. The last section of the chapter looks at productive parents (i.e. parents whose offspring achieve perfect fitness), examining the fitness values of these parents and the operators that influence the final transition to successful solution.

8.2 Search Strategy Options

For consistency with previous metaheuristics each trial is permitted 25000 evaluations of the objective function, consisting of 50 generations of 500 individuals. Four operators are used, Mutation (.01), crossover (.09), duplication (.01) and swap (.01). These values were influenced by previous published re-

search [22] on GE.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length Range	10-100	10-100	10-100	10-100
Population Size	500	500	500	500
Number of Generations	50	50	50	50
Probability of Swap	.01	.01	.01	.01
Probability of Duplication	.01	.01	.01	.01
Probability of Crossover	.90	.90	.90	.90
Probability of Mutation	.01	.01	.01	.01

Table 8.1: Parameters used to configure the Genetic Algorithm.

8.3 Experimental Conditions

The parameters used to configure GA for these trials are presented in Table 8.1. In addition to these *Remainder Stochastic Sampling without replacement* was used as the selection mechanism, with a steady state replacement model used to refine the population at each generation and one point crossover employed as the recombination mechanism.

8.4 Results

Table 8.2 shows the results from the Genetic Algorithm trials. Symbolic Integration proves to be the easiest to solve scoring 100% success, Blocks scores 99.5% and Santa Fe has a score of 81%. With a score of 36% the Genetic Algorithm successfully solves the Symbolic Regression problem following previous failures by Random Search, Hill Climbing and Simulated Annealing.

Problem	Successful Runs
Symbolic Integration	100%
Santa Fe Trail	81%
Blocks	99.5%
Symbolic Regression	36%

Table 8.2: Results from the Genetic Algorithm Trials.

8.5 Productive Parents in GA

Productive parents are parents whose offspring achieve perfect fitness. To understand the relationship between parent and the way in which the child is produced we have analysed the contribution of the operators used. Table 8.3 provides an insight into the contribution of the various GA operators, mutation, crossover, duplication and swap on each of the problems.

Productive parents of Santa Fe solutions are typically of mid to high fitness averaging a score of 43. An examination of the evolution from productive parent to fit child reveals that both crossover and mutation operators occur in the successful transitions.

Table 8.3 also shows the effect of removing each operator in turn for the Santa Fe problem. Removing the mutation operator from GA results in a slight decrease in the success rate, while removal of the crossover operator significantly degrades performance. An examination of the results in Table 8.4 shows that the average fitness value of the productive parent does not change significantly. When the crossover operator is removed the transition from productive parent to fit child is achieved through a combination of mutation and the duplication operator. Removing the duplication operator has no significant impact on performance while removal of the swap operator results in a slight decrease in the success rate.

	Successful Trials			
Operators Used	Santa Fe	Sym Integration	Sym Regression	Blocks
All operators present	81%	100%	36.4%	99.5%
Mutation excluded	76.2%	99%	14.1%	98%
Crossover excluded	18.9%	11%	0.59%	4.7%
Swap excluded	68.3%	100%	34.2%	99.1%
Duplication excluded	82.3%	100%	41.4%	98.6%

Table 8.3: Analysis of Contribution of GA Operators to the Success Rate for Trials.

GA parents for the Symbolic Integration problem are typically of low fitness, indeed 64% of all productive parents map to the same building block $x^2 + x$ which has a fitness of 0.109735. A combination of both mutation and crossover features predominantly in the final steps to the perfectly fit child. The contribution of the mutation and crossover operators to the Symbolic Integration problem are shown in rows two and three of Table 8.4. Removing mutation has no influence of the success rate, however removing the crossover

	Average Fitness			
Operators Used	Santa Fe	Sym Integration	Sym Regression	Blocks
Maximum Fitness	89	1	1	30
All operators present	43	0.0952	0.625	16
Mutation excluded	50	0.1039	0.5549	17
Crossover excluded	44	0.1060	0.383	20
Swap excluded	57	0.099	0.607	17
Duplication excluded	49.7	0.0944	0.630	17

Table 8.4: Analysis of Contribution of GA Operators to Productive Parent's Average Fitness for the Selected Problems.

operator sees the performance drop dramatically falling to a success rate of 11%. The average fitness of the productive parents for this problem shows no significant change (see Table 8.4) when specific operators are excluded.

The Blocks problem shows a dependency on the crossover operator to maintain performance while the removal of the mutation operator has little effect showing no significant drop in the number of successes. The duplication and swap operators appear to have no significant impact on performance. Productive parents for the blocks problem are of mid to high fitness and this level of fitness is maintained when the various operators are excluded.

For the more difficult Symbolic Regression problem mutation features exclusively in the final transition from productive parent to perfectly fit child. Removing mutation sees the performance of GA drop to 14.1%, while removing crossover sees the successes rate fall to less than 1%. This fall in performance is consistent with results seen in previous research [23] [24] which identified the importance of one point crossover in maintaining performance.

Crossover provides more than an explorative search capability, even in

later generations it makes a non destructive contribution by effectively utilises the intrinsic polymorphism of GE's codons as changes in one codon ripple through the expression's sub trees. (see Section 3.2.1 for a discussion on ripple crossover).

An examination of individuals in the later generations of the Symbolic Regression problem shows considerable bloat with the average individual having a length of 185. The average number of genomes required to solve the problem is 42, so crossover has little effect in the transition from productive parent to the final individual, however, it appears to be essential in maintaining the performance of GA.

The removal of the swap operator has no significant impact while removal of the duplication operator actually increases the success rate to 41%. The average fitness of the productive parent remains consistent except for the case where crossover is excluded failing to around half.

8.6 Characteristics of Solutions found by Genetic Algorithms

Table 8.5 provides details of the solutions found by the Genetic Algorithm, showing wrapping featuring in 90% of Santa Fe solutions and 85% of Blocks solutions, consistent with the high level of wrapping seen in these problem using the previous methodologies. Symbolic Integration and Symbolic Regression only uses wrapping in 1% of their solutions. In the case of Symbolic Regression this can be predicted to an extent by the average length of a solution, which is 228 of which 42 are expressed. Significantly, the average number of expressed codons for the Blocks problem is longer at 49 than that required for Symbolic Regression (42), suggesting that the difficulty of

a problem does not necessarily relate to the number of codons in the solution, contrasting with some of the initial results we saw from Random Search which suggested otherwise.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	23	19	30	228
Avg Number of expressed Codons in Solution	13	43	49	42
Avg Number of Solutions featuring Wrapping	1%	90.8%	85%	1%

Table 8.5: Analysis of Features from Solutions found by the Genetic Algorithm.

8.7 Summary

In this chapter we have looked at Genetic Algorithms, the algorithm that has been used in all previously published work on GE. After initial consideration of some of the search strategy options we presented the results of the trials which saw the Symbolic Regression problem solved for the first time with a success rate of 36%. We next turned our attention to productive parents and examined the impact of selectively removing each of the featured operators. Crossover proved to be essential in maintaining performance in all cases while the removal of mutation significantly affected the Symbolic Regression problem. Duplication and swap proved to have no significant impact on the results. Finally we looked at the characteristics of the solutions found by the Genetic Algorithm and established that problem difficulty is not strictly

related to the number of codons required to successfully solve a problem.

Chapter 9

Evolutionary Strategies

9.1 Introduction

This chapter looks at Evolutionary Strategies, the second of three population based methods. Like Genetic Algorithms, it uses both mutation and recombination operators to explore the search space through successive generations. One notable difference between the Genetic Algorithms and Evolutionary Strategies is the the use of a fixed length representation in the latter. The use of fixed length representations is motivated by a desire to be consistent with the commonly used forms of Evolutionary Strategies and the difficulty associated with using variable-length genomes with the established recombination operators (see Section 3.8).

We look initially at the options used to configure the basic algorithm before examining the principle strategy choices of selection and recombination. The results of the main trials are presented with some commentary and analyses of the solutions found by this method.

9.2 Search Strategy Options

For Evolutionary Strategies the population size has been set at 50 (this population size is typical of Evolutionary Strategies [1]), while the number of generations is set to 500, this provides us with 25000 evaluations of the objective function as used in previous trials. The genome length is fixed at 100 for Symbolic Integration, Santa Fe and Blocks. A longer length of 200 was used for Symbolic Regression. An initial *mutation rate* of 12% was chosen with a *mutation rate range* of 25% based on results from preliminary trials. This initial rate of 12% is then mutated for each individual as it evolves with that individual through each generation. The *mutation rate range* value of 25% is the upper limit permissible for the mutation rate after the first generation. So, for example, in the first generation an individual could be subjected to between 0% and 12% mutation and between 0% and 25% on every subsequent generation. A more complete explanation of this process is provided in Section 3.8. The mutation operator operates at the codon level, with the mutation rate expressed as a percentage of the number of codons in a genome, so for example, a figure of 12% means that up to 12% of the codons in a genome could be mutated in the operation.

9.3 Experimental Conditions

Table 9.1 shows the main parameters used to configure Evolutionary Strategies. In order to determine the most suitable selection and recombination methods a number of preliminary trials were performed. These are covered in the next section.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Fixed Genome Length	100	100	100	200
Population Size	50	50	50	50
Number of Generations	500	500	500	500

Table 9.1: Parameters used to Configure Evolutionary Strategies.

9.4 Evolutionary Strategies Selection Methods

As discussed in Section 3.8 there are two preferred methods of selection used in ES, the $(\mu+\lambda)-ES$ (plus) strategy allows the best μ individuals from the union of parents (μ) and offspring (λ) to survive while the (μ, λ) (comma) strategy allows the best (μ) offspring's survive to the next generation.

Evolutionary Strategies also includes a selection of recombination operators (see Section 3.8), discrete recombination involves selecting two parents at random from the population and creating an offspring, which consists of object variables genes drawn with equal probability from both parents, while panmictic (global) intermediate recombination involves selecting one parent at random from the population and creating an offspring through recombination with other randomly selected parents.

To determine the best combination of these options an initial trial of 500 runs was performed, the results of which are shown in Table 9.2. An initial mutation rate of 12% and a mutation range of 25% were used for

these preliminary trials. The *plus* selection strategy which allows the best individuals from the union of parents and offspring to survive has a distinct advantage while intermediate recombination which involves recombination of an individual with a number of other randomly selected parents has a slight advantage over its discrete counter part.

	Selection Strategy			
Problem Type	Comma Intermediate	Comma Discrete	Plus Intermediate	Plus Discrete
Symbolic Integration	25.2%	28.8%	95.4%	95.4%
Santa Fe Trail	17.2%	24.2%	54.2%	51%
Blocks	25.4%	23.8%	99.2%	98.4%
Symbolic Regression	0.2%	0%	25.2%	20.2%

Table 9.2: Comparative Analysis of ES Strategies.

9.5 Impact of Mutation Rate on Success

One of the other key strategy choices with Evolutionary Strategies is the rate of mutation. Two factors influence this in these trials, the initial mutation rate is the starting rate for the individuals in the initial population, the mutation rate range is an upper limit imposed on the mutation rate as it evolves and adapts with the individual.

To evaluate the impact of different values for these parameters a number of trials were conducted on two of the problems, Santa Fe and Symbolic Regression. Table 9.3 shows the results of trials on the Santa Fe and Symbolic Regression problem. The Santa Fe problems shows significant improvement for higher rates of mutation, while on the Symbolic Regression problem there

is a statistically significant difference for mutation ranges less than 25% indicating that there is a minimum level of mutation that is required to maintain a reasonable success rate.

Initial Rate	Range	% Success	
		Santa Fe	Sym Reg
5%	5%	13%	13%
5%	10%	33%	29%
5%	25%	54%	21%
12%	25%	54.2%	25.2%
15%	25%	63%	22%
15%	40%	66%	25%
25%	25%	50%	22

Table 9.3: Comparative Analysis of Mutation Rates and Ranges for Santa Fe and Symbolic Regression

9.6 Results

The final results for the Evolutionary Strategy trials are provided in Table 9.4. All of these trials use intermediate recombination with a *plus* selection strategy. An initial mutation rate of 15% and a mutation rate of 25% have been used.

Symbolic Integration and Blocks have the highest success rates at 95% and 96% respectively. Santa Fe is solved in 63% of the trials and Symbolic Regression has a success rate of 22%. This is only the second metaheuristic to solve the Symbolic Regression problem with Genetic Algorithms previously solving it with a success rate of 36%.

Problem	Successful Runs
Symbolic Integration	95%
Santa Fe Trail	63%
Blocks	96%
Symbolic Regression	25%

Table 9.4: Final Results from Evolutionary Strategies Trials.

9.7 Characteristics of Solutions found by Evolutionary Strategies

Table 9.5 shows the characteristics of the solutions found by Evolutionary Strategies. Because we are using fixed lengths all of the solutions have a length of 100 or more codons (200 in the case of Symbolic Regression), this also helps to account for the figure of 0% for wrapping because the required number of codons for a solution is always less than 100 for these particular problems.

When ES was forced to use wrapping by restricting the maximum genome length to be less than the average solution length it failed to find any solutions for the Symbolic Integration problem. A similar experiment on Santa Fe saw a success rate of 55% even when genome lengths were restricted.

The numbers of expressed codons in a successful solution are largely in line with previous results, again the Blocks problem requires the largest number of codons at 53 with Symbolic Regression requiring 49 on average.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	100	100	100	100
Avg Number of expressed Codons in Solution	18	53	39	49
Percentage of Solutions featuring Wrapping	0%	0%	0%	0%

Table 9.5: Analysis of Characteristics from Solutions found by Evolutionary Strategies.

9.8 Summary

In this chapter we looked at the performance of Evolutionary Strategies, a population based method similar in many respects to Genetic Algorithms. We examined a number of search strategy choices, including selection strategy, recombination operator and mutation rates. We found a *plus* selection strategy and a intermediate recombination operator to be the preferred choice for all of the problems. We also looked at the impact of the mutation rate and found the higher levels of mutation (25%) were more successful on the two problems we selected. Evolutionary strategies has proved itself successful on all four problems, although with lower numbers of successful trials than Genetic Algorithms.

Chapter 10

Evolutionary Programming

10.1 Introduction

Evolutionary Programming is the last of the selected metaheuristics to be evaluated in these trials. Although similar in many respects to Evolutionary Strategies it evolved independently from the work of Fogel, Owens and Walsh in the 1960s. The main difference between the two approaches is Evolutionary Strategies' absence of a recombination operator and its use of a tournament based selection method where an individuals probability of selection is based the number of wins achieved against 10 randomly selected opponents [1]. Like Evolutionary Strategies it uses a fixed length representation.

We look first at the search strategy options and the experimental set-up used in the trials. Some initial runs are performed to assess the impact of the selected strategy variables before the final results are presented. The chapter concludes by looking at some of the characteristics of solutions found by Evolutionary Programming.

10.2 Search Strategy Options

A population size of 50 individuals evolving over 500 generations provides the 25000 evaluations of the objective function used in previous trials. The genome length is fixed at 100 for Symbolic Integration, Santa Fe and Blocks, while a longer length of 200 was used for Symbolic Regression.

The key strategy choice for this method revolves around the selection of suitable strategy variables (see Section 3.7). Two have been chosen for these trials, *mutation rate*, which determines the probability of a codon being mutated, and *mutation bias* which determines where in the genome the mutation can take place.

10.3 Experimental Conditions

Table 10.1 shows the main parameters used to configure Evolutionary Programming. The initial mutation rate has been set at 12% based on the results from this parameter value in the Evolutionary Streategies trials. The *mutation rate range* value is set at 25% (Section 9.2 provides more detail on these parameters).

10.4 Impact of Strategy Variables

A number of preliminary runs were performed to evaluate the impact of the selected strategy variables. The first set uses mutation with no positional bias, that is the mutation can fall anywhere within the entire genome length. The second set uses a strategy variable that limits the mutation of codons to a certain point in the genome. It is expressed as a percentage, where for example 0% means that mutation can take place anywhere in the gemone,

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Fixed Genome Length	100	100	100	200
Population Size	50	50	50	50
Number of Generations	500	500	500	500
Initial Mutation Rate	12%	12%	12%	12%
Mutation Range	25%	25%	25%	25%

Table 10.1: Parameters used to Configure Evolutionary Programming.

50% means that mutation is limited to the second half of the genome and 100% effectively stops any mutation from taking place.

The results show in Table 10.2 shows a significant performance benefit for Symbolic Integration, Santafe, Blocks and Symbolic Regression when mutation without bias is used.

10.5 Results

Table 10.3 summarises the overall results for Evolutionary Programming. These final trials use mutation without positional bias, a mutation rate of 12 and a mutation range of 25. Symbolic Regression is solved with the same success rate of Evolutionary Strategies (25%). Santa Fe is the one problem which sees Evolutionary Programming (73%) outperform Evolutionary Strategies (63%). Scores for the Blocks problem showing no significant difference at around 95%. There is also no significant difference in performance

	Strategy Variables	
Problem	Mutation without Bias	Mutation with Bias
Symbolic Integration	94%	42%
Santa Fe Trail	73%	21%
Blocks	95%	64%
Symbolic Regression	25%	11%

Table 10.2: Analysis of EP Strategy Variables showing Success Rates with and without Mutation Bias

for the Symbolic Integration problem with Evolutionary Strategies scoring 95% and Evolutionary Programming scoring 94%.

Problem	Successful Runs
Symbolic Integration	94%
Santa Fe Trail	73%
Blocks	95%
Symbolic Regression	25%

Table 10.3: Results from Evolutionary Programming Trials.

10.6 Characteristics of Solutions found by Evolutionary Programming

Table 10.4 provides details of the solutions found by Evolutionary Programming A surprising aspect of these results is the emergence of wrapping in some of the solutions. This is curious when one considers that fixed length

genome of length 100 are being used in the case of Santa Fe and Blocks. This length is much longer than the number of codons required to solve the respective problems, however Evolutionary Programming still manages to employ wrapping by finding solutions whose number of expressed codons are in excess of 100. This is contrast to the situation with Evolutionary Strategies in the last chapter which also employed fixed length genomes of similar lengths, yet no wrapping was used in any of the solutions found by that method.

One contributing factor towards the re-emergence of wrapping is that the solutions found by Evolutionary Programming tend to have larger numbers of expressed codons than any of the previous metaheuristics. Santa Fe, for example, uses on average 74 expressed codons as against an average of 46 for all of the previous metaheuristics. An examination of successful solutions shows very high levels of nesting of conditionals in the Santa Fe and Blocks problems. Perhaps the lack of a recombination operator forces Evolutionary Programming toward longer genome lengths in a search for successful solutions.

Feature	Sym Int	Santa Fe	Blocks	Sym Reg
Avg Number of Codons in Solution	100	100	100	200
Avg Number of expressed Codons in Solution	19	74	48	57
Percentage of Solutions using Wrapping	0%	2.79%	46%	1%

Table 10.4: Analysis of Features from Solutions found by Evolutionary Programming.

10.7 Summary

This chapter looked at the last of our selected metaheuristics, Evolutionary Programming. Despite the absence of a recombination operator, the algorithm successfully solved all four problems. Success rates across all of the problems are not significantly different than the scores achieved by Evolutionary Strategies.

A significant aspect of Evolutionary Programming solutions was their length, with the number of expressed codons exceeding those of the other metaheuristics, causing two of the problems to employ wrapping even when fixed length genomes of 100 codons were used.

Chapter 11

Comparison of Methods

11.1 Introduction

This chapter takes a comparative look at the six metaheuristics, Random Search (RS), Hill Climbing (HC), Simulated Annealing (SA), Genetic Algorithms (GA), Evolutionary Strategies (ES) and Evolutionary Programming (EP). We first present a summary of the results, showing success rates for each of the Symbolic Integration, Santa Fe, Blocks and Symbolic Regression problems.

We also attempt to profile aspects of the search space presented to the metaheuristics and summarise some of the characteristics of the problems and their solutions.

The most successful of the metaheuristics, the three population based search strategies, GA, EP and ES are analysed and compared, first comparing GA and ES before turning our attention to EP and GA.

Finally, we look at wrapping and search for characteristics of the problems or aspects of the grammars that might influence it.

11.2 Comparing the results

Table 11.1 provides a summary of success rates for all problems and metaheuristics. GA and ES emerge as the strongest of the search strategies, with the population based methods in general showing a clear advantage over the local search based approaches of HC and SA.

The results indicate that Symbolic Regression has a high degree of difficulty, remaining unsolved by RS, HC and SA. HC would appear to be the least successful of all of the search strategies evaluated, with RS emerging as the most successful of the non-population based search methods.

Metaheuristic	Sym Int	Santa Fe	Blocks	Sym Reg
Random Search	40%	30%	24%	0%
Hill Climbing	8%	15%	17%	0%
Simulated Annealing	11%	21%	13%	0%
Genetic Algorithms	100%	81%	99.5%	36%
Evolutionary Strategies	95%	63%	96%	25%
Evolutionary Programming	94%	73%	95%	25%

Table 11.1: A Summary of Success Rates for all Metaheuristics.

11.3 Characteristics of the Search Space

The results in the previous section show a performance benefit in using population based methods (GA,EP and ES). In order to understand the aspects of the search space that might explain the limitations of the local search methods we profiled the search space for each of the problems.

The profiles for each of the problems consist of all of a successful solution's

immediate neighbours within a hamming distance (i.e. a single codon difference in the genome) of one. The hamming distance of one was chosen as the basis for this profile because it represents the smallest possible unit of change between one solution and another. Only expressed codons are changed, the unexpressed tail is removed from the genome before the neighbourhood is generated. Once all neighbours have been generated the unexpressed tail is re-attached to each of the individuals in the neighbourhood. This re-attached tail allows them utilise the ripple effect to form new solutions. It should be noted that neighbours in this context are not ancestors of successful solutions, rather they are generated from successful solutions using the method outlined above. The objective here is to provide some insight into the sensitivity of successful genomes to change.

The plots shown are a composite of all solutions from all of the trials. This data is also presented in summary form in Table 11.2 where we show the percentage fall off in fitness of the immediate neighbourhood. So, for example, in the case of Santa Fe, 22.30% of the neighbours also achieve maximum fitness, while 0.49% have a score that is within 20% of maximum fitness.

11.3.1 Santa Fe Trail

The plot for Santa Fe shown in Figure 11.1 shows a sharp fall of in fitness from a plateau of maximum fitness. An examination of all of the Santa Fe solutions found during the trials shows that, on average, 22.3% of neighbours (see Table 11.2) of a Santa Fe solution will also have perfect fitness. Of these 22.3% perfectly fit neighbours, 20% are identical mapping to the same solution and following the same sequence in selecting productions.

The average number of expressed codons in a solution for Santa Fe is

	Percentage Drop in Fitness (0 - 50%)					
Problem	0%	10%	20%	30%	40%	50%
Santa Fe	22.3%	0.11%	0.49%	0.15%	0.36%	2.08%
Sym Integration	7.09%	0.02%	0.0%	0.0%	0.0%	0.0%
Sym Regression	4.39%	1.38%	2.09%	1.17%	2.16%	1.91%
Blocks Stacking	33.0%	0.55%	0.96%	12.64%	2.61%	6.55%
	Percentage Drop in Fitness (50 - 100%)					
Problem	50%	60%	70%	80%	90%	100%
Santa Fe	2.08%	0.72%	4.21%	8.82%	35.62%	25.15%
Sym Integration	0.0%	0.02%	0.04%	2.36%	83.38%	7.04%
Sym Regression	1.91%	3.43%	5.56%	7.75%	56.48%	13.69%
Blocks Stacking	6.55%	11.58%	5.82%	0.23%	0.0%	25.50%

Table 11.2: Percentage fall off in Fitness of Immediate Neighbours for all Problems.

51, which is much longer than the nine required for the minimal successful solution. One of the structures that tends to create some of the longer solutions is the multiple nesting of the $if(trail.food\_ahead())$ conditional.

The profile of the Santa Fe neighbourhood might help explain why the local search approaches experiences difficulty, within the neighbourhood of a solution there are points of very low fitness, with no progressive slope for the algorithms to climb. An approach that attempts to fine tune a solution through gradual exploration of the immediate neighbourhood is always likely to perform poorly.

Santa Fe shows a reduction in performance when *Genetic Code Degeneracy* is reduced by restricting the range of codon values used to the maximum

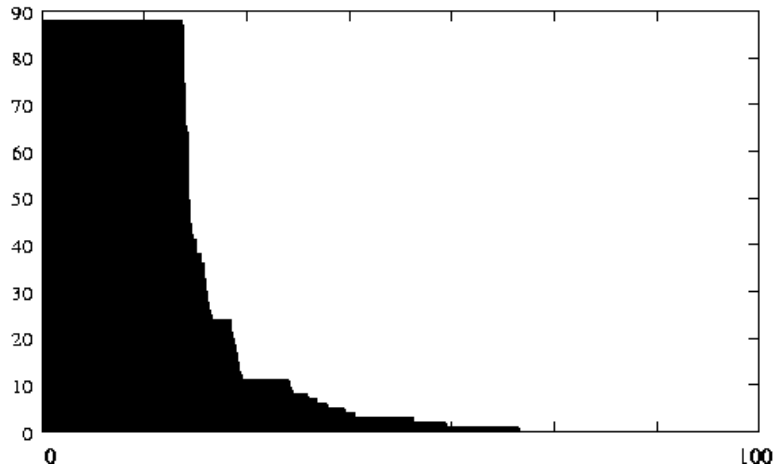

Figure 11.1: Neighbourhood of a Solution for The Santa Fe Trail Problem. The graph shows the immediate neighbourhood of solutions. The Y-axis shows the fitness while the X axis displays the neighbours sorted in descending order.

number of production rules in the grammar (i.e from 256 to 3). A trial of 1000 runs using Random Search with a maximum codon value of two saw the performance drop to 4% from the original success rate of 30%. Increasing the maximum codon value by one to three causes the performance recover to the original 30%.

This sensitivity is possibly a consequence of the simple modulo function (see Section 2.3) used by GE. This rule selection function is degenerate allowing many codon values to map to the same choice of rules. One of the consequences of this is a linkage between production rules belonging to different non-terminal symbols. This linkage, which is strongly influenced by layout and sequencing of the rules introduces a bias into the search process [11].

Degeneracy also acts toward the preservation of the functionality of the phenotype while allowing unrestricted exploration of the genotypic search space as evidenced by increases in the number of invalid individuals when it

is removed [31] [30].

11.3.2 Symbolic Integration

An analysis of the immediate neighbourhood of solutions for Symbolic Integration (see Figure 11.2) shows a different profile to that of Santa Fe, we again see fitness fall off sharply from a peak of maximum fitness, however the plateau of maximum fitness is greatly reduced. Just over 7% of the immediate neighbours of solutions retain maximum fitness, indeed Table 11.2 shows that over 83% of all neighbours have values that have fallen by 90%. The implications of this is that area of the global maxima (that is, the correct answer) is significantly smaller than in the Santa Fe problem, so relatively small changes in individuals in this neighbourhood are considerably less likely to result in a global maxima. The effects of this can be seen in table 11.1 where the performance of HC and SA have both dropped.

Symbolic Integration shows a less pronounced effect when the range of the codon value is restricted to reduce the impact of *Genetic Code Degeneracy* . We see the success rate for Random Search fall from 40% to 27% when the maximum value of the codon is reduced from 255 to two.

11.3.3 Blocks

Figure 11.3 shows a rather different profile to that of the other problems, we see a more gradual slope to maximum fitness. This profile would tend to suggest that approaches like HC and SA should perform strongly relative to RS, and while we do see less of a performance gap between RS and HC/SA, RS still scores better. This may partly be explained by the scoring scheme used in the Blocks problem (see Section 4.5), which makes it quite easy to

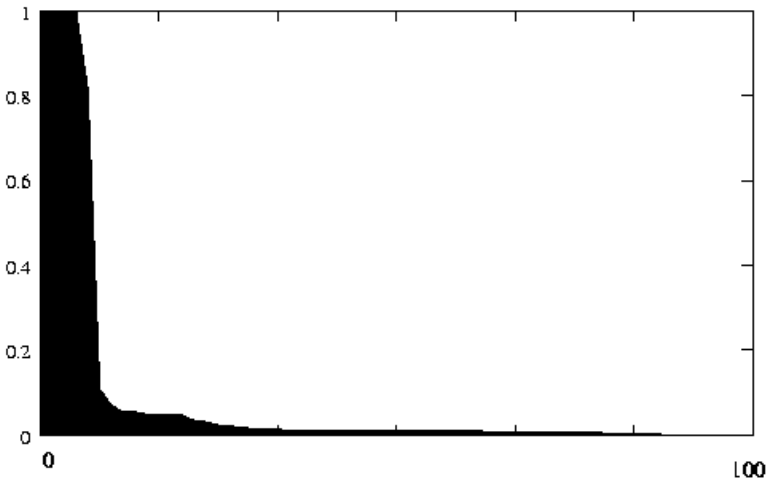

Figure 11.2: Neighbourhood of the Solution for The Symbolic Integration Problem. The graph shows the immediate neighbourhood of solutions. The Y-axis shows the fitness while the X axis displays the neighbours sorted in descending order.

score a relatively high fitness level for solutions that fall short of correctly stacking the blocks.

When the range of codon values is reduced in the Blocks problem we see similar results to that of Santa Fe with the success rate for Random Search based trials fall from 24% to 10%.

Solutions for the Blocks problems tend to be quite long requiring 57 codons on average, which is longer than the number of codons required for any of the other problems and also longer than the shortest found for this problem which required 20 codons. The aspects of the grammar that contribute to this long length are redundant sequences of the negation operator (!) and some nesting of the *Do Until* construct.

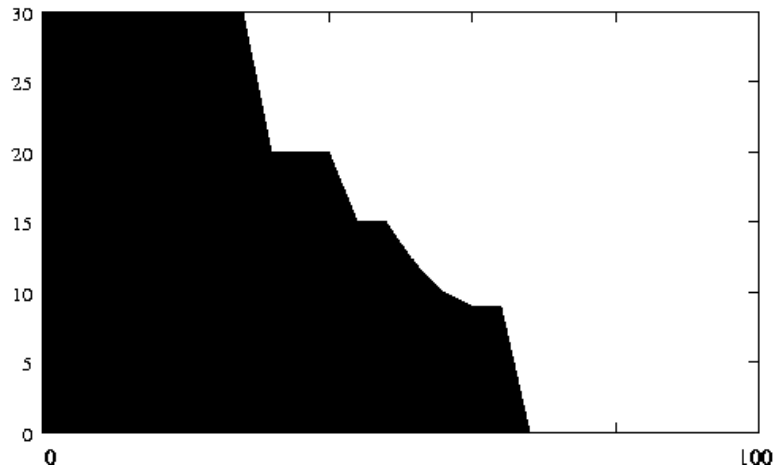

Figure 11.3: Neighbourhood of a Solution for The Blocks Stacking Problem. The graph shows the immediate neighbourhood of solutions. The Y-axis shows the fitness while the X axis displays the neighbours sorted in descending order.

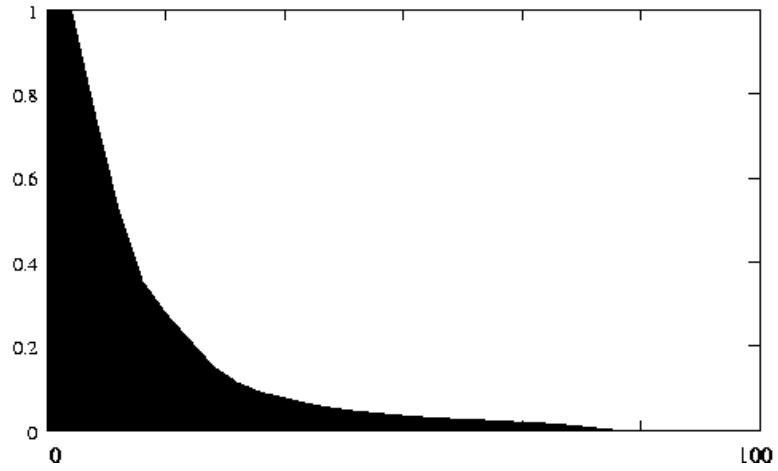

Figure 11.4: Neighbourhood of a solution for The Symbolic Regression Problem. The graph shows the immediate neighbourhood of solutions. The Y-axis shows the fitness while the X axis displays the neighbours sorted in descending order.

11.3.4 Symbolic Regression

The neighbourhood of a Symbolic Regression problem is similar to that of Symbolic Integration, showing a sharp isolated peak with a more gradual yet pronounced fall off in fitness as shown in Figure 11.4. Solutions for Symbolic Regression require 49 codons on average, which is less than the number required for Blocks or Santa fe.

ES was used to evaluate Symbolic Regression's sensitivity to changes in the range of codon values. We see evidence of a strong dependency to the original range of values as suggested by the drop in success rates falling from 25% to 1% when the maximum genome value is set to two. This contrasts with the results from the Symbolic Integration which shares the same grammar.

11.4 Analysis of Solution Lengths

The average numbers of expressed codons in a solution are presented in Table 11.3 showing Blocks as the problem requiring the largest number at 57, Symbolic Integration requires fewest at 16 while Santafe and Symbolic Regression require 51 and 49 respectively. The solution lengths for Blocks and Santa Fe are quite long when one considers the shortest solutions found for these problems, for example Blocks can be solved with a genome length of 20 and Santa Fe only requires 9 codons for success. The factors contributing to this tendency toward longer genome lengths will be explored further in Section 11.8.1.

Metaheuristic	Sym Int	Santa Fe	Blocks	Sym Reg
Random Search	14	46	69	n/a
Hill Climbing	15	47	77	n/a
Simulated Annealing	19	48	91	n/a
Genetic Algorithms	13	43	49	42
Evolutionary Strategies	18	53	39	49
Evolutionary Programming	19	74	71	57
Average	16	51	66	49

Table 11.3: A Summary of Successful Solution sizes for all Metaheuristics.

11.5 Comparing ES, EP and GA

While ES and GA share many similarities, with both approaches employing both mutation and recombination operators there are a number of significant differences in their approach. The selection strategy used by ES in these trials was the plus strategy (see Section 3.8), which selects the best individuals from the union of parents and offspring to survive. Selection in this GA consists of *remainder stochastic sampling without replacement* with steady state populations. The mutation operator in GA is based on the probability of mutating an individual bit in a codon whereas ES uses mutation at the codon level using a mutation rate that evolves as a strategy variable from generation to generation.

ES uses fixed length genomes while GA uses variable length genomes that can grow in length under the influence of the crossover and duplication operators. Finally, the population sizes differs with GA using a larger population size of 500 against a typical ES population size of 50.

EP and GA also have a number of significant differences, with EP using

a population size of 50 against 500 for GA. EP also uses a fixed length genome, while GA uses variable length genomes. The selection methods differ in that GA uses *remainder stochastic Sampling without replacement* with steady state populations while EP uses a *tournament based* selection strategy to select individuals from a combined population of parents and offspring.

The mutation operator in GA is based on the probability of mutating an individual bit in a codon whereas EP uses mutation at the codon level using a mutation rate that evolves as a strategy variable from generation to generation.

Rows three to six of Table 11.1 show the results from the original ES, EP and GA trials, while the results from these methods are reasonably close, overall, GA shows a statistically significant advantage over ES and EP. This is most apparent on the Santa Fe and Symbolic Regression problems.

11.6 Fixed and Variable length Gemones on EP and GA

Metaheuristic	Sym Int	Santa Fe	Blocks	Sym Reg
GA (pop 500)				
no crossover	11%	18.9%	4.7%	0.59%
EP (pop 50)	94%	73%	95%	11%

Table 11.4: Results from a Comparative Analysis of GA with no crossover and EP.

In Section 8.5 we looked at the performance of GA with the crossover

operator removed. Table 11.4 shows the results of EP alongside those of GA with crossover removed, revealing a significant difference in the performance of EP over GA.

While GA and EP have many differences in their structure, in this scenario they both operate without the benefit of crossover, relying on mutation to create new individuals. One of the characteristics that distinguishes them is the use of fixed length genomes by EP and variable length genomes by GA.

The normal operation of the GA algorithm uses randomly generated lengths of genomes to create the initial population. EP differs in this respect by using fixed length genomes of a specific pre-determined size. Introducing fixed length genomes into GA sees a significant improvement in its performance when the crossover operator is omitted. Conversely, the introduction of variable length genomes to EP does not adversely influence its success rate, indeed Symbolic Regression shows a significant increase in performance from 11% to 20%. These results are shown in Table 11.7, with rows one and two showing the original success rates and three and four showing the changes when fixed length genomes are introduced to GA and variable length genomes are used in EP.

11.7 Experimental Conditions

The parameters used to configure fixed length GA for these trials are presented in Table 11.5. Variable length EP used the parameters presented in Table 11.6. The variation in EP individuals is established at the initial creation of the population. Because EP uses only mutation the length of the individuals does not change beyond the initial generation.

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length	100	100	100	200
Population Size	500	500	500	500
Number of Generations	50	50	50	50
Probability of Swap	.01	.01	.01	.01
Probability of Duplication	.01	.01	.01	.01
Probability of Crossover	0	0	0	0
Probability of Mutation	.01	.01	.01	.01

Table 11.5: Parameters used to configure the Fixed Length Genetic Algorithm.

11.8 Wrapping

One curious aspect of these results in the prominence of wrapping in solving some problems and the degree to which the proportion of solutions using wrapping varies between metaheuristics. Symbolic Regression has little dependence on wrapping, appearing only when GA is used, while Santa Fe and Blocks show high proportions of wrapping in solutions with Santa Fe using wrapping on average in 46% of all solutions while Blocks has an even higher proportion at 52%. On average Symbolic Integration uses wrapping in only 0.16% of solutions, indeed wrapping does not appear in any Symbolic Integration solutions found by RS, ES or EP.

We created a set of trials for the Symbolic Integration in order to deter-

Parameter	Problems			
	Sym Int	Santa Fe	Blocks	Sym Reg
Number of Trials	1000	1000	1000	1000
Number of Objective Function Evaluations	25000	25000	25000	25000
Initial Genome Length Range	10-100	10-100	10-100	10-100
Population Size	50	50	50	50
Number of Generations	500	500	500	500

Table 11.6: Parameters used to Configure Evolutionary Programming.

mine if the absence of wrapping in Symbolic Integration is due to the fact that the required number of expressed genomes is low (16) relative to the length of solutions typically generated by the various search strategies. The initial genome length was set to 10, which is six codons shorter than the 15 codons required on average for a successful Symbolic Integration solution. As a control we ran a similar trial for Santa Fe with the genome length set at 38, also six shorter than the average number required for a solution. The selection of these particular lengths was arbitrary, the only criteria being that GE would be forced to wrap in the majority of cases.

Random Search was used as the search mechanism for these trials in which Santa Fe scored a 29% success rate, not significantly different than the score achieved in the original Random Search trials (30%). Symbolic Integration, however, was successful in none of the 500 runs.

The results of these trials indicate that the absence of wrapping in the case of Symbolic Integration is not due to the short length of genome required to solve this particular problem, when the problem is forced to use wrapping it fails, which is contrasted with Santa Fe which readily copes with the short

Metaheuristic	Sym Int	Santa Fe	Blocks	Sym Reg
GA no crossover	11%	18.9%	4.7%	0.59%
EP	94%	73%	95%	11%
GA no crossover Fixed Length	74% (+63%)	54% (+35.1%)	46% (+41.3%)	33% (+32.4%)
EP Variable Length	94% (0%)	62% (-11%)	91% (-4%)	20% (+9%)

Table 11.7: Analysis of the Impact of Fixed and Variable Length Genomes on GA without crossover and EP.

Genome lengths by employing wrapping to create solutions. These results would seem to suggest that the tendency to use wrapping is linked to the nature of the problem and/or its associated grammar.

11.8.1 Grammars and Wrapping

Table 11.8 provides a summary of the proportion of wrapping associated with each of the problems. Symbolic Integration and Symbolic Regression show little, if any, dependency on wrapping, while Blocks and Santa Fe have a significant amount of solutions that use wrapping. Symbolic Integrations and Symbolic Regression both use the same grammar, which provides some fundamental building blocks $(x, +, -, *, sin)$ to build the target expressions $f(x) = Sin(x) + x^2 + x$ and $x^4 + x^3 + x^2 + x$. The solutions for these expressions show a high degree of repetition, often combining long sequences of the combination $x * x$, the repetitive nature of this could be one of the

factors that contributes against wrapping appearing in solutions.

Metaheuristic	Sym Int	Santa Fe	Blocks	Sym Reg
Random Search	0%	43%	41%	n/a
Hill Climbing	0%	80%	70%	n/a
Simulated Annealing	0%	50%	66%	n/a
Genetic Algorithms	1%	91%	85%	1%
Evolutionary Strategies	0%	0%	0%	0%
Evolutionary Programming	0%	25%	46%	1%
Average	0.16%	48%	51%	1%

Table 11.8: Proportion of Wrapping appearing in Successful Solutions for all Metaheuristics.

11.8.2 Impact of Conditionals on Wrapping

Another distinction between the grammars is the presence of conditionals,

```
DO  <code> UNTIL <code> ODU
```

in the case of Blocks and

```
if(food_ahead()){ <line> else <line>}
```

for the Santa Fe problem. The solutions that use wrapping typically have multiple nested loops using these constructs. These multiply nested loops are often redundant in that they add nothing to the actual logical flow of the program but they may provide a means of consuming codons in a way that allows wrapping provide solutions. A example of this nesting from a Santa Fe solution is shown below.

```
if(trail.food_ahead())
   {
   if(trail.food_ahead())
      {
      if(trail.food_ahead())
         {
         if(trail.food_ahead())
            {
            if(trail.food_ahead())
               {
               trail.move()
               }
            else
               {
               trail.move()
               }
            }
         else
         ................
```

In order to determine if the presence of conditionals in a grammar increases the probability of wrapping we introduced a conditional into a grammar used for solving the Symbolic Integration problem. This grammar was a simplified version of the grammar presented in Section 4.3.

```
<expr>          :: ==  <expr><op><expr> | <pre-op>(<expr>) | <var>
<var>           :: == X
<op>            :: == + | - | / | *
<pre-op>        :: == sin | cos
```

In this simplified form the order of some of the production rules were deliberately changed to first determine if this in itself had an impact. We first performed a 1000 runs with the new grammar without any conditionals present. We used RS as the search mechanism and found that wrapping did not appear in any of the solutions found. When the problem was forced to use wrapping by using short genome lengths it failed on every occasion to find a solution.

We then introduced a simple conditional into the grammar. The conditional took the form of

```
IF (1) THEN <expr> ELSE <expr> ENDIF
```

This had the effect of allowing solutions to be enclosed in a conditional construct that always defaulted to the first expression by virtue of the fact that the *IF* statement was always true. The complete grammar took the following form:

```
<expr>          :: ==  <expr><op><expr>
                     | <pre-op>(<expr>)
                     | <var>
                     | <control>
<control>       :: IF ( 1 ) THEN  <expr>  ELSE  <expr>  ENDIF
<var>           :: == X
<op>            :: == + | - | / | *
<pre-op>        :: == sin | cos
```

When we again conducted a 1000 runs with RS using this new grammar we found that wrapping appeared in 5% of the solutions found. When wrapping was forced on the method by using short genome lengths it managed a

success rate of 14%. While these results would suggest a possible relationship between wrapping and the presence of conditionals in a grammar much more research is required to fully understand the nature of the relationship.

It is also important to remember that the operators also influence the probability of wrapping appearing in solutions. This is most evident in the contrasting results of EP and ES, where we saw wrapping appear in EP but not in ES. Significantly, both methods used fixed length genome that were sufficiently longer than the required number of codons for a successful solution. These results would suggest that something in the EP approach, perhaps the absence of recombination (which is present in ES) forced the method toward longer genome lengths utilising wrapping to solve the problem.

There is probably no simple single explanation for the emergence of wrapping in some problems. A detailed cross comparison of the codons of successful solutions shows no obvious patterns in either those that use wrapping and those that do not. The results of the research on the Symbolic Integration grammar, which saw wrapping emerge when we included a simple conditional in the grammar, suggests that an area for future research would be an analysis of the correlation between the presence of these type of constructs and the probability of wrapping. This could be supported by the selection of problems and the construction of grammars that are likely to have both condition constructs and high levels of repetition.

11.9 Summary

In this chapter we have taken a comparative look at the results featured in the previous chapters. These results show a strong performance by the popu-

lation based search strategies like GA, EP and ES, while the local search approach of HC and SA have limited success. Curiously, RS performs strongly on three of the problems, out-scoring HC and SA.

We looked also at the problems Symbolic Integration, Santa Fe, Symbolic Regression and Blocks, attempting to offer some insight into the search space in the immediate neighbourhood of solutions. The profiles of three of the problems suggested a terrain that would be difficult for local search metaheuristics like HC and SA.

GA, ES and EP were contrasted and compared, looking in particular at the impact of fixed and variable length genomes on GA and EP.

Finally, we looked at wrapping and attempted to understand why it appeared so prominently in two of the problems. We found that the presence of wrapping is not due solely to the length of genome solutions as demonstrated by Symbolic Integration which failed to find any solutions when forced to use wrapping. As a basis for some future work we suggested research that looked at particular constructs in the grammars in particular, loops based on conditionals and large amounts of repetition in solutions.

Chapter 12

Conclusions and Further Research

12.1 Summary

The primary goal of this thesis was to evaluate and understand the performance and behavior of Grammatical Evolution (GE) by using a number of different search methodologies. In the course of this research, six different metaheuristics were evaluated Genetic Algorithms (GA), Evolutionary Strategies (ES), Evolutionary Programming (EP), Random Search (RS), Hill Climbing (HC) and Simulated Annealing (SA). These evaluations were conducted on four benchmark GP problems, Symbolic Integration, Santa Fe Trail, Blocks and Symbolic Regression.

Chapter two introduced the mechanics of GE and explained much of the associated terminology giving particular emphasis to concepts associated with language grammars. Concepts like genetic code degeneracy, neutral mutation, and expressed and un-expressed genes, which are referenced throughout the text, were introduced.

In chapter three we introduced the chosen metaheuristics, HC, SA, GA, RS, GA, ES and EP, presenting each with a discussion on the origin of the method, the key strategy choices, operators and a detailed step by step guide to the algorithms involved.

The problem domains were introduced in chapter four with an explanation of the objective of the problems and a presentation in Backaus Naur Form of each of the grammars used to attempt to solve the problems.

Each of the six chapters from five through to ten focused on a particular metaheuristic. A similar pattern was used for each looking first at the search strategy options, then considering the parameters used to configure the algorithms, presentation of the results and some discussion on aspects of the metaheuristic that could be refined or tuned to improve the success rate.

In chapter eleven we looked comparatively at the results, contrasting the performance of the different metaheuristics on each of the problems, looking at the nature of the search space encountered by the search strategies and looking at characteristics of the successful solutions found. The latter part of this chapter focused on more detailed analysis of some of the more successful metaheuristics GA, ES and EP. Finally, we looked at the wrapping and tried to understand aspects of the problems and their grammars that might contribute to it prominence in some results and its absence in others.

12.2 Conclusions

The relationship between problems, grammars and GE is a complex one. While this research has shown that many different search strategies can be used successfully with GE, it is difficult to extract universal learnings that guarantee improvements across all classes of problems. We can, however,

say that population based methods have a distinct advantage as evidenced from the performance of EP, ES and GA. These metaheuristics clearly outperformed the local search based approaches of HC and SA. Indeed, any attempts to try and improve the performance of HC and SA through variation in the search strategy options for these methods yielded little improvement. While the performance of the local search approaches was disappointing, the performance of RS was surprisingly strong, significantly out performing HC and SA on the three problems that it managed to solve.

GA was the best search strategy to solve the Symbolic Regression problem, successfully solving it in 36% of attempts. EP and ES also solved this problem at lower success rates. Overall, GA was more successful on the chosen problems, with little to differentiate the performances of the EP and ES.

We analaysed the contribution of the various GA operators in an effort to understand some of the factors that contributed toward its success and found that crossover was key to the performance of the algorithm with the removal of mutation, duplication and swap having lower levels of influence on success rates. A significant difference in the emergence of successful solutions in GA as opposed to HC and SA, was the fact that GA solutions typically emerged from mid to high fitness parents while successful solutions from the local search approach emerged from areas of low fitness.

While operator by operator analysis revealed that the contribution of crossover was seen as key to the performance of GA, a comparative analysis of GA and EP revealed that the performance of GA without crossover can be significantly boosted by the introduction of fixed length genomes. It is also significant that the form of intermediate recombination favoured by ES in our trials is potentially much more disruptive an operator than the one-

point crossover used by GA. This, coupled with the fact that EP used no recombination at all, would tend to suggest that to some degree all of the operators in isolation are dispensable to some degree, the key to success being the global search approach offered by the population based nature of these methods.

ES had similar levels of performance to GA, with Symbolic Regression being the only problem in which we saw any significant difference. Intermediate recombination and a *plus* selection strategy was clearly the most successful form of ES and an analysis of successful solutions from ES showed no wrapping being used in any of the successful solutions.

EP introduced a second strategy variable that controlled where in the genome string the mutation would occur, however, this proved counter productive and best results were achieved with the standard form of adaptive mutation that we had used previously in ES. Despite the absence of a recombination operator, which had proved essential in maintaining GA performance, results for EP were not significantly worse that those of GA and ES. One of the most curious features of the EP results was the re-emergence of wrapping in the Santa Fe and Blocks problems despite the fact that the length of genome presented to the search strategies was longer than the number of codons required for a successful solution.

Our attempts to profile the problem search spaces and our evidence from the various search strategies used suggest complex, noisy, discontinuous multimodal landscapes. These landscapes prove difficult for the deterministic gradient following approaches of HC and SA, while the global approach of GA, EP and ES cope significantly better. With the exception of the Blocks problem, we saw solutions to our selected problems exist as sharp isolated peaks of maximum fitness, and further evidence from our use of RS to sample

the search space suggest that some of the problems have a high density of such peaks.

One of the most significant things to emerge from this research was the recognition of GE's ability to work successfully and seamlessly with a broad range of different search approaches. This establishes it as a powerful and flexible mechanism for the automatic generation of grammar based code and programs.

12.3 Future Directions

An area requiring a more complete analysis is the relationship between grammars and wrapping. In this research we have tried to understand some of the factors that influence wrapping. The relationship between the various factors that contribute to its emergence in certain problems is complex. We saw in two of the problems, Santa Fe and Blocks, a tendency toward solutions that are considerably longer than the minimum number of codons required. This is facilitated primarily by nesting of conditionals and by sequences of redundant code. We did establish that wrapping is not purely related to the required number of codons in a given solution. This was evident when we tried to force Symbolic Integration to use wrapping by presenting it with short genome lengths only to see it fail at every attempt, unable to complete a solution through wrapping.

We have attempted to shed some light on some of the factors that contribute toward the emergence of wrapping, however we believe that the most fruitful area for further research into this behaviour lies in the selection and construction of specific problem grammars that use constructs like conditionals and repetition. Additionally the design of the grammar is proving

increasingly important with evidence that layout and sequencing are strongly influencing linkage within the grammar effecting production selection bias.

The performance of RS on at least three of the problems brings into question their suitability as bench mark problems. While the population based methods have proved more successful on these particular problems there is no guarantee that these results will hold true for more complex problems. Future studies in both GE and GP would be better served by moving away from these particular problems.

While much work was done in the optimisation and in various search strategy options were introduced in this for the three most successful population based methods much work remains in the analysis and optimisation of these methods. Approaches like EP and ES offer the opportunity of strategy variables evolving with the emerging solutions, as our knowledge of the inner workings of GE continues to grow these strategy variables could be used to exploit this knowledge by engineering the search in a particular direction, some modest attempts at this have already been attempted in this research. The analysis of a population based approach to SA and HC would also be an interesting avenue of investigation given their dependence on a good starting point and the proven success of population based algorithms.

Appendix A

Other Publications by the Author

With Conor Ryan, (2002). An Investigation into the Use of Different Search Strategies with Grammatical Evolution. In Foster, J.A., Lutton, E., Miller, J.F., Ryan, C., Tettamanzi, A., (editors): *Lecture Notes in Computer Science 2278, Proceedings of the Fifth European Conference on Genetic Programming*, pages 123-130, Kinsale, Ireland, April 2002. Springer-Verlag.

Bibliography

[1] Bäck T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford University Press.

[2] Carpenter, G., Grossberg, S., Markuzon, N., Reynolds, J., and Rosen, D. (1992). Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. in *IEEE Trans. Neural Networks., Volume 3*, pages 698-713, Sept. 1992.

[3] Chellapilla, K., Fogel, D. B. (1997). Exploring Self Adaptive Methods to Improve the Efficiency of Generating Approximate Solutions to Traveling Salesman Problems Using Evolutionary Programming. In Angeline, P. J., Reynolds, R. G., McDonnell, J. R., Eberhart, R. C., (editors): *Lecture Notes in Computer Science 1213, Evolutionary Programming VI, 6th International Conference*, pages 361-372. EP97, Indianapolis, Indiana, USA, April 13-16, 1997, Springer 1997, ISBN 3-540-62788-X

[4] Duvivier, D., Preux, P., Talbi, E.-G. (1996). Climbing up NP-hard hills. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., Wasniewski, J., (editors): *Lecture Notes in Computer Science 2328, Proceedings of the 4th International Conference on Parallel Processing Applied Mathematics (PPAM '01)*, pages 35-41, Springer-Verlag, 2002 ISBN 3-540-43792-4

[5] Eleseth, G.D, Baumgardner, K. D. (1995). *Principles of Modern Genetics.* West Publishing Company.

[6] Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky (ed.), Advances in Neural Information Processing Systems 2. Morgan Kaufmann.

[7] Glover, G., Lagunas, M. (1995). *Tabu Search. In Modern Heuristic Techniques for Combinational Problems*, chapter 3. McGraw-Hill Book Company, Berkshire, 1995.

[8] Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley.

[9] Holland, J.H. (1992). *Adaptation in Natural and Artifical Systems.* MIT Press.

[10] Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M. and Vladan, B. (2002). Ripple Crossover In Miller, J., Tomassini, M., Lanzi, P.L, Ryan, C., Andrea, G.B., Langdon, W.B., (editors): *Lecture Notes in Computer Science 2038, Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2001*, pages 74-86. Lake Como (Milan), Italy, 18-20 April 2001. Springer-Verlag.

[11] Keijzer, M, O'Neill, M., Ryan, C., and Cattolico, M. (2002). Grammatical Evolution Rules: The Mod and the Bucket Rule. In Foster, J.A., Lutton, E., Miller, J.F., Ryan, C., Tettamanzi, A., (editors): *Lecture Notes in Computer Science 2278, Proceedings of the Fifth European Conference on Genetic Programming*, pages 123-130, Kinsale, Ireland, April 2002. Springer-Verlag.

[12] Kimura, M. (1983). *The Neutral Theory of Molecular Evolution.* Cambridge University Press.

[13] Kirkpatrick, S., Gerlatt, C. D. Jr., and Vecchi, M.P. (1983). Optimization by Simulated Annealing, In *Science Volume 220, No. 4598*, pages 671-680, 1983.

[14] Koza, J. (1992). *Genetic Programming.* MIT Press.

[15] Koza, J. (1994). *Genetic Programming II: Automatic Discovery of Reusable Program.* MIT Press.

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

[17] Lang, K. J. and Witbrock, M. J. (1988). Learning to tell two spirals apart. In Touretzk, D.S., Hinton, G.E., Sejnowski, T.J., (editors): *Proceedings of the 1988 Connectionists Model Summer School*, pages 52-59. Morgan Kaufmann.

[18] Langdon, W. B., Poli R. (1998). Why Ants are Hard. In Koza, J. R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoym Dorigo, Marco, Fogel, D. B., Garzon, Max, H., Goldberg, D. E., Iba, Hitoshi, and Riolo, Rick L. (editors): *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193-201, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin. San Francisco, CA. Morgan Kaufmann.

[19] Langdon, W. B. (1999). Size Fair and Homologous Tree Genetic Programming Crossovers. In In Banzhaf, W., Daida, J., Eiben, A. E., Gar-

zon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., (editors): *Proceedings of the Genetic and Evolutionary Computation Conference*, Volume 2, pages 1092-1097, Orlando, Florida, USA,13-17 July 1999. Morgan Kaufmann.

[20] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1958). Equations of State Calculations by Fast Computing Machines, In *Journal of Chemical Physics, Volume 21*, pages 1087-1091, 1958.

[21] Mitchell, M. and Holland, J. H. (1993). When will a Genetic Algorithm Outperform Hill Climbing? In Cowan, J., Tesauro, G., and Alspector, J., (editors): *Advances in Neural Information Processing Systems*, pages 51-58, San Francisco, CA., 1994. Morgan Kauffman.

[22] O'Neill, M., Ryan, C. (2001). Grammatical Evolution. *IEEE Trans. Evolutionary Computation*, Volume 5 No. 4, August 2001

[23] O'Neill, M., Ryan, C., Keijzer, M and Cattolico, M. (2001). Crossover in Grammatical Evolution: The Search Continues. In Miller, J., Tomassini, M., Lanzi, P. L., Ryan, C., Andrea, G. B., Langdon, W. B., (editors): *Lecture Notes in Computer Science 2038, Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2001*, pages 337-347. Lake Como (Milan), Italy, 18-20 April 2001. Springer-Verlag.

[24] O'Neill, M., Ryan, C. (2000). Crossover in Grammatical Evolution: A Smooth Operator? In Banzhaf, W., Fogarty, T., Langdon, W. B., Miller, J., Nordin, P., Poli, R., (editors): *Proceedings of the Third European Workshop on Genetic Programming 2000*, pages 149-162, Edinburgh, Scotland, UK, April 2000

[25] O'Neill, M., Ryan, C. (1999). Evolving Multi-Line Compilable C Code. In Poli, R., Nordin, P., Langdon, W. B., Fogarty, T. C., (editors): *In Proceedings of the Second European Workshop on Genetic Programming 1999*, pages 83, Gteborg, Sweden, May 26-27, 1999.

[26] O'Neill, M., Brabazon, T., Ryan, C., Collins, J.J (2001). Developing a Market Timing System using Grammatical Evolution. In Beyer, H., Cantu-Paz, E., Goldberg, D., Spector, L., Whitley, D., (editors): *Proceedings of GECCO 2001*, Morgan Kaufmann 2001 ISBN 1558607749

[27] O'Neill, M., Brabazon, T., Ryan, C., Collins, J.J (2001). Evolving Market Index Trading Rules using Grammatical Evolution. In Boers, E.J.W., Cagnoni, S., Gottlieb, J., Hart, E., Lanzi, P.L., Raidl, G.R., Smith, R.E., Tijink, H., (editors): In *Lecture Notes in Computer Science 2037, Applications of Evolutionary Computing*, pages 343-35, Springer-Verlag.

[28] O'Neill, M., Keijzer, M. and Cattolico, M., Ryan, C. (2003). Crossover in Grammatical Evolution. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (editors): In *Lecture Notes in Computer Science 2610, Proceedings of the Sixth European Conference, EuroGP 2003*, Springer-Verlag.

[29] O'Neill, M., Ryan, C. (1999). Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond. In Floreano, D, Nicoud, J.D , Mondada, F., (editors): *Lecture Notes in Computer Science 1674, Proceedings of the Fifth European Conference on Artifical Life, ECAL99*, Lausanne, Switzerland, September 13-17, 1999. Springer-Verlag.

[30] O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution. PhD thesis, University of Limerick. M0005921LK

[31] Ryan, C., O'Neill, M. (2003). *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers.