Establishing Mechanisms for Self-Adaptation in Genetic Programming

Thomas D. Griffiths

Doctor of Philosophy

Aston University

June 2019

Aston University
Establishing Mechanisms for Self-Adaptation in Genetic Programming
Doctor of Philosophy Thomas David Griffiths 2019

Thesis Abstract

It has long been a desire of computer scientists to develop a computer system that is able to learn and improve without being explicitly programmed to do so. The idea of software that is able to analyse, update and alter itself has been discussed.

The thesis is structured as follows: *Firstly*, we refine and improve the Tartarus problem, proposing it as a benchmark problem for use in GP. *Secondly*, we establish a mechanism for incorporating self-adaptation into a GP system in order to increase the performance of candidate solutions. *Finally*, we explore the impact of a fitness bias, inspired by the Dunning-Kruger effect, on the robustness of a GP system.

The on-the-fly adaptation of parameter values at runtime can lead to improvements in performance. Self-adaptation aims at biasing the distribution of individuals in a population towards more appropriate and effective areas of the search space.
Therefore, we propose, outline and evaluate a novel self-adaptive mechanism favouring a continuous opportunity for modifications to be made during an execution, as-and-when they are deemed to be appropriate. This creates a more flexible parameter modification approach, leading to an increase in solution performance: leading to an approximate 15% and a 10% increase for the Tartarus and Santa-Fe problems respectively.

Robustness is often referred to as a characteristic of a candidate solution whose performance is not diminished despite perturbations in environmental parameters or constraints. A solution that does not lose utility or performance quality under these changes is said to be *robust*.
The Dunning-Kruger Effect (DK) is a form of cognitive bias observed in populations, first described by psychologists Dunning and Kruger in 1999: individuals with a low level of ability mistakenly over-estimate their performance and conversely, individuals with a high level of ability will often under-estimate their performance.
We propose that that the introduction of a DK style bias into the fitness distribution of the population will enable a system to maintain a higher level of population diversity over time.

Key words: Self-Adaptation - Dunning-Kruger - GP - Robustness - Diversity

Acknowledgements

I would like to express my gratitude to my supervisor Anikó Ekárt for her continued encouragement, contribution and mentorship; without which, this thesis would likely never have been finished.

My thanks go out to my friends and colleagues in the ALICE research lab and wider Computer Science department for their guidance.

I would like to extend my love and appreciation for my partner Ciara, and my dearest family Charlotte, David, Helene, Ian and Pippa.

Finally, I would like to thank my grandparents, Ruth & Bryan, for everything they have done for me throughout my life, supporting me when I truly needed it the most.

Thank-you.

Contents

Publications arising from this thesis

Conference Proceedings

[**1**] **Short Paper** – *T.D. Griffiths.*
"Increasing Genetic Programming Robustness using Simulated Dunning-Kruger Effect"
In. *Genetic and Evolutionary Computation Conference* ***GECCO 2019***
GECCO 2019 Companion Proceedings.
pp. 340–341, ACM 2019.

[**2**] **Full Paper** – *T.D. Griffiths, and A. Ekárt.*
"Self-Adaptive Crossover in Genetic Programming: The Case of the Tartarus Problem"
In. *15th International Conference on Parallel Problem Solving from Nature* ***PPSN 2018***
Proceedings Part I | vol. 11101 — *Lecture Notes in Computer Science*
pp. 236–246, Springer 2018.

[**3**] **Full Paper** – *T.D. Griffiths, and A. Ekárt.*
"Improving the Tartarus Problem as a Benchmark in Genetic Programming"
In. *20th European Conference on Genetic Programming* ***EuroGP 2017***
Proceedings | vol. 10196 — *Lecture Notes in Computer Science*
pp. 278–293, Springer 2017.

[**4**] **Short Paper** – *T.D. Griffiths, and A. Ekárt.*
"Improving the Effectiveness of Genetic Programming using Continuous Self-Adaptation"
In. *2nd International Symposium on Artificial Life and Intelligent Agents* ***ALIA 2016***
Post Proceedings | vol. 732 — *Communications in Computer and Information Science*
pp. 97–102, Springer 2016.

Workshop Contributions

[**1**] *T.D. Griffiths, and C.M. Barnes.*
"Self-Adaptive Task Separation in the Tartarus Problem"
At. 3rd Workshop on Self-Awareness in Cyber-Physical Systems – *SelPhyS*
Workshop Presentation. 2018. – url `http://thomas-david.uk/docs/w2a`

[**2**] *C.M. Barnes, T.D. Griffiths, A. Ekart and P. Lewis*
"A Family of Environments for Exploring Social Self-Awareness"
At. 3rd Workshop on Self-Awareness in Cyber-Physical Systems – *SelPhyS*
Workshop Presentation. 2018. – url `http://thomas-david.uk/docs/w2b`

[**3**] *T.D. Griffiths.*
"Self-Adaptive Crossover in Genetic Programming"
At. 1st Funcollective on Evolution and Learning in Socio-Technical Systems
Workshop Presentation. 2018. – url `http://thomas-david.uk/docs/w3`

List of Abbreviations

APM - Adaptive Parameter Modification
CGP - Cartesian Genetic Programming
DK - Dunning–Kruger (effect)
EC - Evolutionary Computing
GA - Genetic Algorithm
GGP - Grammar-Based Genetic Programming
GP - Genetic Programming
LGP - Linear Genetic Programming
SAGP - Self-Adaptive Genetic Programming
SAPM - Self-Adaptive Parameter Modification
SMCGP - Self-Modifying Cartesian Genetic Programming
TGP - Tree-Based Genetic Programming
TP - Tartarus Problem

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

‘ Certain books seem to have been written. not in order to afford us any instruction, but merely for the purpose of letting us know that their authors know something. ’

Johanne Wolfgang von Goethe

Chapter Contents

1.1 Scenario

It has long been a desire of computer scientists to develop a computer system that is able to learn and improve without being explicitly programmed to do so. The idea of software that is able to analyse, update and alter itself has been discussed. The computational research field of Genetic Programming (GP) emerged three decades ago, as a method towards achieving automatic programming [1, 2, 3].

In GP, an initial population of candidate solutions, members of the set of possible solutions, is created and iteratively updated. The 'fitness' – success at solving the desired task, of each candidate solution is evaluated, and a new population of candidate solutions is subsequently stochastically generated.
This is done by removing the less successful solutions and generating new solutions from the more successful solutions remaining in the population. In biological terms, the population of candidate solutions is subjected to both random mutation and the pressures of *natural selection* [4]; often referred to as the *survival of the fittest*.

In order to assess the performance of chosen approaches, a range of standardised test problems are often used. These problems, referred to as benchmarks, are utilised in order to illuminate features of an algorithm and evaluate performance. In this thesis we propose a range of improvements [5] to the Tartarus Problem (TP) [6], updating the state evaluation mechanism and providing guidance on tuning the difficulty of instances. Moreover, we outline the characteristics of the Tartarus problem, suggesting its use as a suitable benchmark problem [7].

In GP, the on-the-fly adaptation of parameter values at runtime can lead to improvements in performance. This *self-adaptation* of parameter values aims at biasing the distribution of individuals in a population towards more appropriate and effective areas of the search space. This is generally achieved by means of setting and adjusting control parameters [8], expressed as *population size*, *recombination probability* or *mutation rate*, to name a few.

The process of optimising the self-adaptive bias is itself a dynamic problem, as a set of control parameters deemed optimal at the start, or during an execution, may be unsuitable at a later stage. One approach to triggering these modifications is to utilise a fixed, pre-determined set of rules or time interval. However, over time, this can lead to an increase in ineffectual adaptations.
In this thesis we propose and outline a novel self-adaptive mechanism: favouring a continuous opportunity for modification during execution, as-and-when they are deemed appropriate. Creating this more flexible parameter modification approach leads to an increase in solution performance of approximately 15% and 10% for the Tartarus and Santa-Fe problems, respectively.

Robustness is often referred to as a characteristic of a candidate solution whose performance is not diminished despite perturbations in environmental parameters or constraints. A solution that does not lose utility or performance quality under these changes is said to be *robust*. The Dunning-Kruger Effect (DK) is a form of cognitive bias observed in populations, first described by psychologists Dunning and Kruger in 1999: individuals with a low level of ability mistakenly

over-estimate their performance and conversely, individuals with a high level of ability will often under-estimate their performance.

In this thesis we propose that the introduction of a DK style bias into the fitness evaluation of the population will enable a GP system to maintain a higher level of population diversity over time: increasing the robustness of the population to changes in Tartarus problem instances. This is achieved by means of modifying the fitness scores of individuals based on their relative performance, in a manner similar to DK. The individuals will have their *actual* true fitness score modified, returning their new *reported* fitness score, leading to an approximate 10% increase in the diversity present in the GP population.

1.2 Overarching Research Questions

This thesis is concerned with the following overarching research questions:

RQ1 Can the Tartarus problem be modified and improved, in order to satisfy the desirable benchmark characteristics, outlined by White et al. [7]?

RQ2 Is it possible to parameterise a genetic programming system by incorporating self-adaption into the evolutionary processes, and does allowing the system to trigger parameter modifications *as-and-when* they are required provide a greater benefit than the canonical method of triggering at discrete intervals?

RQ3 Will the introduction of a simulated Dunning–Kruger effect lead to an increase in the robustness of a genetic programming system?

1.3 Thesis Contributions

The main contributions realised by this thesis are as follows:

- The Tartarus Problem is presented as a benchmark problem, presenting an improved evaluation mechanism with a finer level of granularity. Guidance is provided for tuning the difficulty of generated Tartarus instances.
- A novel self-adaptive crossover operator is presented for use in a GP system. Providing a continuous opportunity for parameter modifications to be made at runtime, leading to an increase in solution performance.
- A novel fitness bias, inspired by the cognitive bias observed in the Dunning–Kruger effect is implemented in a GP system. As a result, an increase in the level of population diversity and solution robustness was observed.

1.4 Thesis Overview

The remainder of the thesis is structured as follows:

Chapter 2 provides a broad introduction to Genetic Programming, introducing the processes of *Selection, Recombination* and *Mutation*, present in a canonical GP system. A selection of different GP representations are presented, with the structural characteristics of each contrasted and compared.

Chapter 3 introduces the assumption that many of the benchmark problems that are currently utilised in GP literature are no longer fit for purpose. A set of *desirable characteristics* for benchmark problems is introduced, alongside an analysis of the suitability of benchmark problems.

Chapter 4 addresses **RQ1**, proposing an updated version of the Tartarus Problem (TP) as a benchmark problem for use in GP. Several facets of the problem have been modified and updated, including the extended parameterisation of instance generation, coupled with guidance on *tuning the difficulty* of generated instances. An improved, more granular method of conducting Tartarus *state evaluations* is also presented. The suitability of the updated problem is assessed and evaluated against the desirable benchmark characteristics.

Chapter 5 focuses on **RQ2**, introducing parameter modification approaches, providing a comparative taxonomy for *deterministic*, *adaptive* and *self-adaptive* techniques. A novel self-adaptive crossover operator for use in GP is presented, providing a continuous opportunity for parameter modifications at runtime. The self-adaptive crossover operator is assessed across two case-studies: firstly, the Tartarus problem and secondly, the Santa-Fe problem, with an approximate increase in solution performance of 15% and 10%, respectively.

Chapter 6 is concerned with **RQ3**, exploring the links between diversity and robustness in a GP population. A novel method for increasing the diversity of a GP population inspired by the Dunning-Kruger effect is introduced. The impact on the diversity and robustness of the GP population is tested on Tartarus problem instances. The analysis shown that it is possible to increase the level of diversity in a population by approximately 10%, leading to an increase in robustness.

Chapter 7 presents the conclusions, summarising the results of the contributions obtained throughout the thesis, followed by the future direction of the research.

Chapter 2

Genetic Programming:

Introduction

‘ You know we are on the wrong track altogether. . .

We must not think of the things
we could do with,

but only the things
we cannot do without. ’

George *a character in* Three Men in a Boat

Jerome K. Jerome

Chapter Contents

2.1 Genetic Programming

Genetic Programming is an evolutionary computing (EC) technique [9], a form of nature inspired computing [10, 11], championed by Koza [12, 13]. As with other EC methods, in a GP system an initial population of candidate solutions is stochastically generated and the performance of each solution is evaluated by use of a 'fitness function' [14]. At each generation, the GP system executes three distinct steps, as shown in Figure 2.1:

Selection, the least successful individuals are removed from the population and the 'parents' of the new population are chosen.

Recombination, the chosen 'parents' are combined to create a new population of candidate solutions, also known as *crossover*.

Mutation, small changes are stochastically applied to individuals within the population.

Source: Griffiths, T.D., and Ekárt, A. [15]

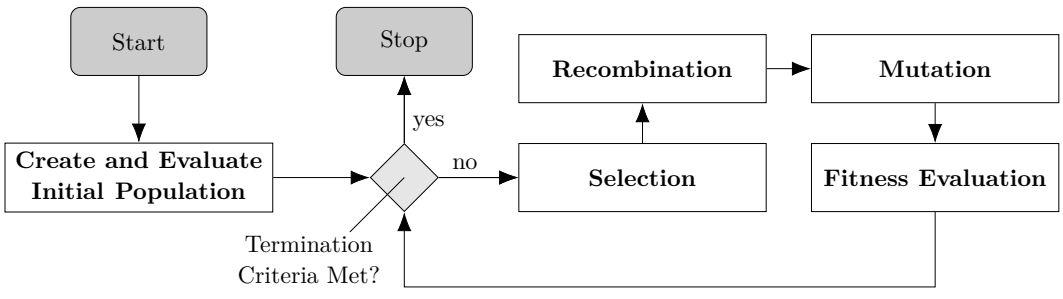


Figure 2.1: An Example GP System
Outlining the relationship between the three distinct steps of a GP system, *selection, recombination, mutation* and the *fitness evaluation*.

2.1.1 Fitness Evaluation

At the end of every generation, each candidate solution in the population is assessed and their performance evaluated by the fitness function [16]. The fitness function is responsible for measuring the progress of each candidate solution in respect to solving the problem task.

Many fitness functions are based on a simple distance metric between the desired outcome and the outcome of the candidate solution, often conceptualised as the *error* of a solution [17]. However depending on the problem, more complex fitness measures can be used [18, 14].

It is possible to combine several measures in order alter the level of evolutionary pressure on a population. One common approach is to combine the *error* of a candidate solution and the *size* of the candidate solution. This can lead to a fitness function which rewards solutions which are not just successful but also small in size. This can be a useful tool when used in conjunction with other approaches in an attempt to reduce the rate of growth in the size of GP candidate solutions, known as *bloat* [12, 19, 20].

2.1.2 Genetic Programming Representations

There are many ways in which GP solutions can be structured and represented, with varying levels of constraints and complexity present in each representation. The canonical and most widely used GP representation, is *tree-based* GP [12], where the candidate solution is represented by a tree-like data structure, along-

side the widely used *linear* GP [21], where the solution is represented as a linear string of instructions [12]. More recent GP representations include *Cartesian* GP [22], where the solution is represented as interconnected graph. Additionally, representations exist that utilise grammars [23], widely grouped together as *grammatical* GP [24].

Historically, many GP systems were represented using tree-based or linear GP, however cartesian GP and more complex representations that allow for self-modification and adaptation are now widely used [25].

2.1.2.1 Tree-Based GP

The earliest and most widely used GP representation is *tree-based* GP (TGP) [12]. In TGP the solutions are encoded as a tree-like data structure, an example of which is shown in Figure 2.2. The solutions can be modified by altering the structure and layout of the tree, by swapping, adding or pruning sub-trees, or by altering the values of the nodes within the tree.

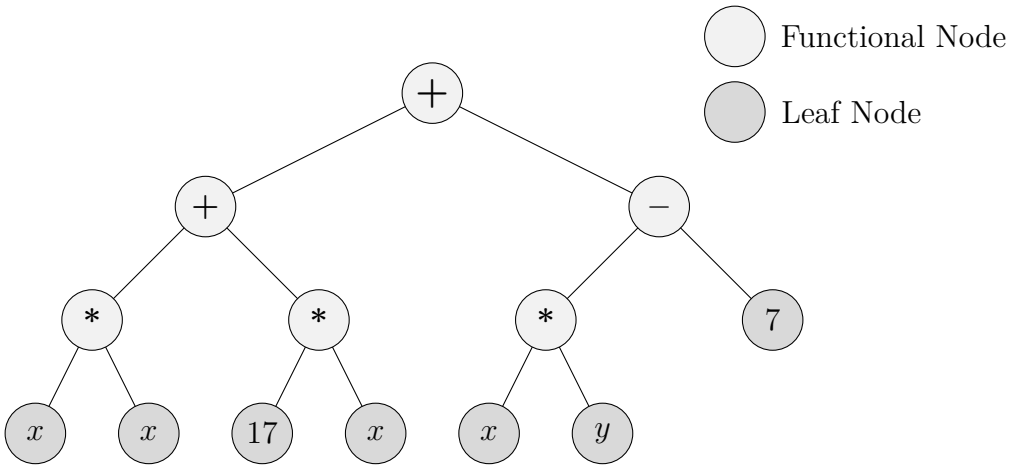


Figure 2.2: The equation $x^2 + 17x + xy - 7$ illustrated in the style of the Tree-based GP Representation

The inner nodes of the tree structure contain functional instructions, while the outer *leaf* nodes contain input variables or constant values.

Due to the hierarchical nature of the tree-like structure utilised in TGP, any change in node values or placement has the potential to radically change the execution order of the tree [26]. The closer a node is to the root of the tree, the more radical the potential impact from modification, either by recombination or mutation. It can be expected that in a tree-based structure that nodes towards the bottom of the tree, on the lower levels, are modified more frequently.[1]

2.1.2.2 Linear GP

Another commonly used GP representation is known as linear GP (LGP) [27]. In LGP the solution is encoded as a linear structure, often referred to as a *genome*, containing a number of functional instructions, input values and constants, referred to individually as *chromosomes*. An example LGP solution is shown in Figure 2.3. The solutions can be modified by altering the structural order of the chromosomes, adding and removing chromosomes[2] or altering the values of the chromosomes.

x	*	x	+	17	*	x	+	x	*	y	−	7

Figure 2.3: The equation $x^2 + 17x + xy - 7$ Illustrated in the style of the Linear GP Representation

[1] It is possible to take precautions in tree-based GP systems in order to manipulate the rate of such modifications through biasing of the operator hit rate.

[2] It is common in GP systems utilising a linear representation for the maximal length of the solution to be fixed, referred to as fixed-length LGP. In these cases the addition of new instructions would not be permitted once the maximum length is reached.

The influence of any change in a linear structure, such as those utilised by LGP, can be expected to follow the linear order in which the instructions are executed. A modification made towards the end of the structure will create less of an impact than a change at the start of the structure [21]. However, unlike tree-based structures, in linear GP it can be expected that changes of all sizes occur equally frequently.

2.1.2.3 Cartesian GP

A more recent GP representation is cartesian GP (CGP) [22, 28]. In CGP the solutions are encoded into a graph-like structure, often conceptualised as a genome similarly to LGP. However, unlike LGP the chromosomes in a CGP genome are not necessarily executed in a linear sequential fashion, as they are encoded as an interconnected graph, many chromosomes are semantically linked to, and potentially have an impact on, other chromosomes elsewhere in the genome [29].

2.1.2.4 Grammar-Based GP

Grammar-based GP (GGP) [24, 23], or *Grammatical Evolution* (GE), as it is widely referred, is used to describe a number of approaches whereby the solutions generally are encoded in a linear structure in a similar manner to LGP, they are then translated into an executable program by use of a grammar [30].

2.1.3 Selection

The selection operator is responsible for selecting individuals which are to be used as parents for the next generation. It is important that the mechanism by which the selection is made is able to effectively differentiate between individuals that

possess beneficial characteristics leading to an increase in solution performance, and those that do not.

As with many Evolutionary Computation approaches, The selection operator utilised in a GP system, probabilistically selects individuals based on their fitness score. That is to say that, individuals with greater fitness scores are more likely to be chosen, compared to individuals with lesser fitness scores.

One widely-used operator is known as **Tournament Selection** (Section 2.1.3.2.), alongside **Roulette Wheel Selection** (Section 2.1.3.3.) and other approaches such as **Truncation Selection** (Section 2.1.3.4.) [27].

In practice however, any selection operator, mechanism or approach that can effectively select individuals from the population that contain the desired characteristics, can be used in a GP system.

2.1.3.1 Selection Pressure

Selection pressure is a term used to describe the property of a mechanism in terms of how strongly it differentiates between good and bad individuals [31].

A mechanism which has a high selection pressure would strongly favour individuals with a higher fitness score. Approaches which favour individuals based solely on their raw performance, at the expense of other metrics and measurements are said to be *greedy* operators.

However, a mechanism which has a lower selection pressure, would place less preference on individuals with a higher fitness score, and in comparison is likely to select a wider and more diverse range of individuals from the population.

2.1.3.2 Tournament Selection

Tournament Selection is a style of selection operator used in GP [32, 33]. In tournament selection a subset of individuals of size k is randomly chosen from population P. The individuals within this subset, the tournament, are compared against each other, with the best individual being chosen as a parent. An example of which is shown in Figure 2.4

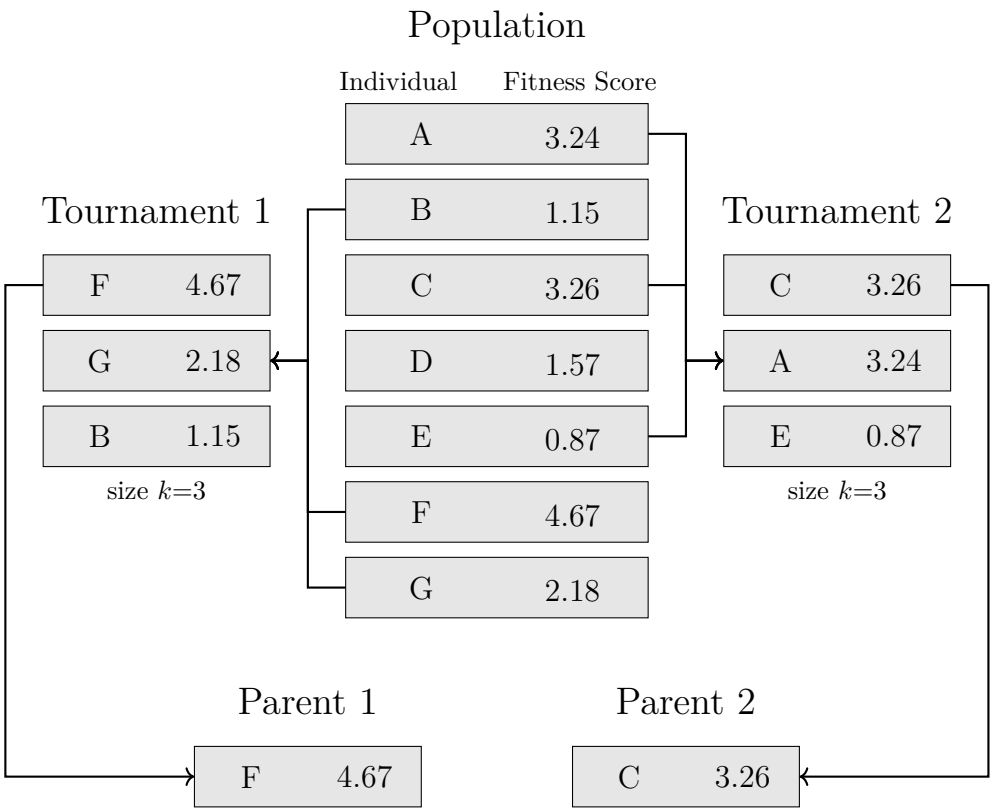


Figure 2.4: Tournament Selection: Showing the selection of parents from the population using two random tournaments of size k=3

In canonical tree-based GP two parents are required, therefore two separate tournaments will take place to select the parents.

Highlights of Tournament Selection

The tournament selection operator is concerned with selecting the best individual from each tournament, it is not concerned with or influenced by the magnitude of the difference.

For example, in Figure 2.4 – *Tournament 1*, the difference between the individuals marked F and G is large, at 2.49. However, in *Tournament 2* the difference between the individuals marked C and A is of a much smaller magnitude, at just 0.02. Thus demonstrating the ability of the tournament selection mechanism to select the individual with the highest fitness score from the tournament, regardless of the magnitude of difference between the individuals.

It is possible to influence the selection pressure placed on the tournament selection operator by altering the tournament size k. For example, a tournament of size $k = 1$ is equivalent to randomly selecting an individual from the population, effectively removing the selection pressure entirely. As the size of k increases, the number of individuals from the population included in each tournament as a proportion of the entire population becomes greater, increasing the selection pressure.

At the point $k = P$, the selection pressure is considered to be at its maximum value – all individuals present in the population P is included in the tournament, with the highest fitness individual is guaranteed to be selected every time.
The act of changing the value of k, in the range $1 - P$, directly and measurably impacts the selection pressure the tournament selection operator exerts on the population.

2.1.3.3 Roulette Wheel Selection

Fitness Proportionate Selection, more commonly referred to as Roulette Wheel Selection, is a style of selection operator used in GP. In fitness proportionate

selection the probability of an individual being selected is directly proportional to the fitness score of the individual [34].

As shown in Figure 2.5, the individuals with a higher fitness score, A and C have a higher probability of being selected, compared to individuals with a lower fitness score, B and D.

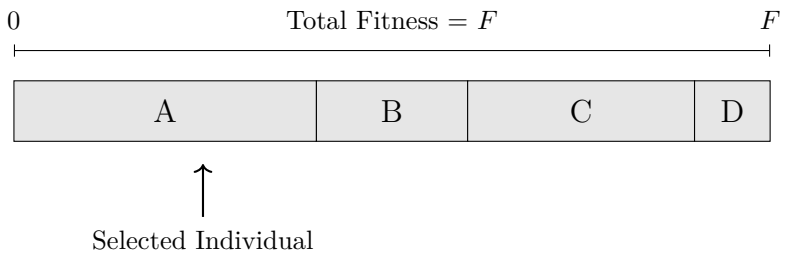


Figure 2.5: Traditional Linear Fitness Proportionate Selection
The likelihood of an individual x being selected is directly proportional to its fitness score.

As the fitness score of an individual increases the probability of it being selected increases proportionally, relative to the other individuals in the population, as shown in Equation (2.1).

$$p_x = \frac{f_x}{\sum_{i=1}^{n} f_i} \quad , \tag{2.1}$$

where p_x is the probability of individual x being chosen, f_x is the fitness score of individual x and n is the number of individuals present in the population.

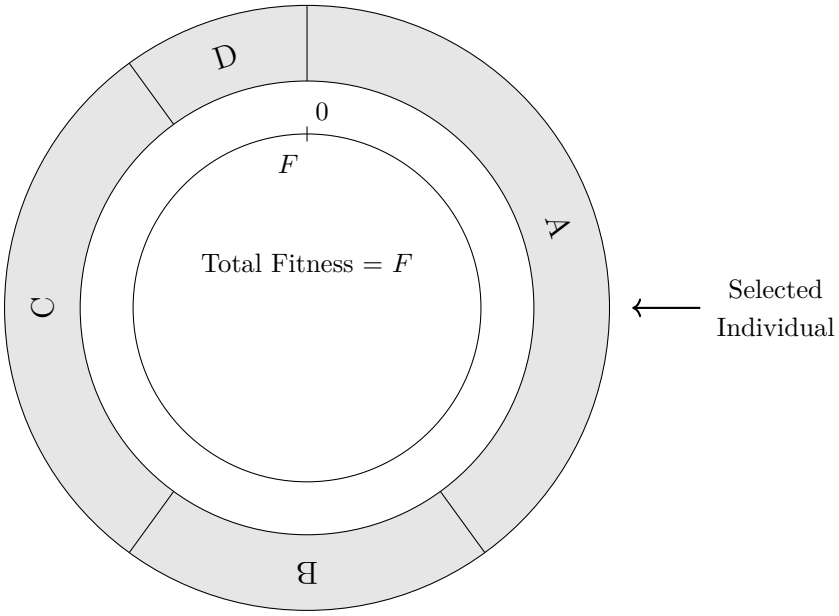


Figure 2.6: Roulette–Wheel Fitness Proportionate Selection
The likelihood of an individual x being selected is directly proportional to its fitness score.

Fitness proportionate selection is commonly referred to as roulette wheel selection, alluding to the often used metaphor of a casino roulette wheel, where the bins on the roulette wheel are sized according to the fitness score of an individual, as shown in Figure 2.6.

Highlights of Roulette Wheel Selection

The fitness proportionate operator is designed to probabilistically select individuals based on their fitness scores, with individuals with a higher fitness score being more likely to be selected.

However, in contrast to tournament selection, the magnitude of the difference between individuals has a large impact on the outcome of fitness proportionate selection.

For example in Figure 2.5, the individual marked A with the highest fitness score, is much more likely to be chosen in comparison to the individual marked D with the lowest fitness score. However it should be noted that again, in contrast to tournament selection, in fitness proportionate selection the sum of all probabilities for each individual being selected is equal to 1.

Therefore, due to the fixed stochastic nature of fitness proportionate selection, although unlikely an occurrence, there is a possibility that the individual marked D with the lowest fitness score in the population, may still be chosen as a parent. This highlights the fact that in fitness proportionate selection, even individuals with low fitness scores may, on occasion, be chosen as parents and contribute to future generations, thus potentially having an impact on the future diversity of individuals within the population[3].

It is for this reason that it is not possible to directly alter the selection pressure the fitness proportionate selection operator exerts on the population.

2.1.3.4 Truncation Selection

Truncation selection is a simple selection operator used in GP. In truncation selection the individuals within a population are ordered according to their fitness score and a proportion p of the fittest individuals are selected.

It is possible to directly alter the selection pressure of truncation selection by changing the size of the proportion value p.

As the individuals in the population are sorted according to their fitness scores,

[3]The link between GP selection operators and the overall diversity of a population is discussed in greater detail in Chapter 6.

reducing p would lead to an increase in selection pressure and a fitter and less diverse range of individuals would be selected.

Alternatively, increasing p would lead to a decrease in selection pressure, leading to a more diverse range of individuals being selected from the population. Truncation selection was explored and analysed in detail by the *distributed breeder genetic algorithm* (DBGA) [35].

2.1.4 Recombination

The recombination operator, often referred to as *crossover*, is responsible for creating new individuals using the individuals chosen previously by the selection operator. We shall refer to the individuals chosen by the selection operator as the *parents* and the newly generated individuals as the *offspring*.

As with the selection operator there are several different established approaches to recombination in GP. The goal of any recombination operator is to take information from the two parent individuals Pi_1 and Pi_2 and *combine* it together in some manner to create a set of new offspring individuals $Oi_1...Oi_n$.

Many recombination operators create either 1 or 2 children, however in practice it is possible to create any number of children as long as the desired overall population size is maintained [12].

In canonical recombination operators, a *crossover point* is chosen, and the two parent individuals Pi_1 and Pi_2 are divided at the selected point. The resultant pieces are then recombined together to generate offspring which contain some information from Pi_1 and some information from Pi_2.

For example in tree-based GP, a chosen crossover point, a node, would be chosen for both Pi_1 and Pi_2. The nodes beneath the chosen crossover points, the *sub-trees* would be transferred between the parents, with the sub-tree from Pi_1 being interchanged for the sub-tree from Pi_2 and vice versa, creating new offspring individuals with a combination of nodes from the both parents.

It is possible to classify recombination operators by considering certain properties of interest which are present in the operators, these properties are *respect*, *assortment* and *transmission*.

2.1.4.1 Respectful Recombination

A recombination operator is considered to be *respectful* if and only if, it generates children which contain *all* of the features that are common across both parents. Therefore, it can be postulated that if the parents are identical to each other, the children generated will be identical to the parents also, this property is referred to as *purity*.

Although it is possible for an operator which is not generally considered *respectful* to create children which exhibit *purity*, all *respectful* operators are implictly *pure* in nature.

2.1.4.2 Assorted Recombination

While the property of *respect* represents the exploitative side of the recombination process, the property of *assortment* represents the exploratory side. An operator is said to be *properly assorting* if it is able to produce *any* combination of features taken from both parents.

Some operators may take several generations or several runs of the recombination operator to create a combination which covers all features of both parents, these are known as *weakly assorted* operators.

On the other hand, operators that are able to combine *any* features of both parents in one single execution are referred to as *strongly assorted* operators.

2.1.4.3 Transmitting Recombination

The final property of interest in recombination operators is the property of *transmission* [27]. Transmission is possibly the most important property of operators, an operator is said to be *transmitting* if every feature in the child can be found in the parents. That is to say that nothing new has been added to the child, it is a combination of features that were present in the parents.

It is possible however to have operators that introduce new features to children which were not present in the parents. These operators, known as *non-transmitting* operators, are often referred to as being responsible for the introduction of *implicit mutation* in the recombination process. This is due to the fact that it is possible for them to introduce features into the new population that are not present in the current population.

2.1.5 Mutation

The final step in the GP system is *mutation* [36]. Unlike the *implicit mutation* which is present in some recombination operators, most notably *transmission* operators, the mutation being carried out in this step is *explicit mutation*. That

it to say that it is done with the express purpose of causing realisable change in the population of individuals.

2.1.5.1 Checked Mutation

Mutation can be divided into two main categories, *safe* and *unsafe* mutation. *Safe* mutation is where the mutation being carried out is checked, ensuring that the resultant individual is still valid.

Unsafe mutation on the other hand does not perform and checks when the mutation is applied. The act of checking for validity is computationally expensive to perform and should be taken into consideration when choosing a mutation operator.

Depending on the GP representation that has been chosen, different types of mutation operators can be used. Some representations are implicitly valid in nature, and there is no need to use a *safe* mutation operator, as this would be a waste of computational effort.

For example, canonical *tree-based GP* is always valid due to the ways in which the trees are constructed, and subsequently mutated. *Linear GP* however does not share this characteristic, and using an *unsafe* mutation operator on these individuals has the potential to create invalid outcomes, so it is necessary to use a *safe* operator.

Chapter 3

Benchmark Problems in Genetic Programming

‘ Never ignore, refuse to see, what may be thought against your thought. ’

Friedrich Nietzche

Chapter Contents

3.1 Benchmark Problems

A benchmark problem is a standardised problem, primarily used for evaluating the performance of a solution. Solutions from different algorithmic methodologies and approaches can be assessed, with the resultant performance comparable [37]. Some benchmarks are utilised in order to illuminate and highlight features of an algorithm. Real-world benchmarks are created in order to reflect real-world problems - allowing for a simplified problem to created, abstracting away some of the complexity whilst still providing valuable insight.

Benchmark problems for automatic programming have been generally classified by McDermott et al, into the following groups [38]:
Symbolic Regression, **Classification**, **Predictive Modelling** and finally, **Route-Finding**, upon which we will primarily concentrate

Traditionally, benchmark problems have been utilised, in an attempt to answer two primary questions [39]:

- Which algorithmic approach results in the best performance on a given task?
- Which algorithmic approach should be used for a given real-world problem?

In order to understand the first question, one must first consider what it means to be the 'best' in the context of performance benchmarking, and begin to enquire as to whether this has any bearing on approaching the second question – the solving of real-world problems.

Empirical studies have attempted to show the effectiveness of many algorithmic approaches. However, these experimental methods of comparison on a given benchmark problem have several drawbacks [40]. One flaw with the traditional empirical evaluation of algorithmic performance is that the resultant conclusions rely just as heavily on the problem being utilised as they do on the algorithmic approach being tested. This often leads to the situation where a tailor made solution performs excellently on one particular benchmark, but the performance does not generalise or transfer well to other similar benchmark problems, known as over-fitting [41].

3.1.1 Symbolic Regression

Symbolic regression problems utilise a form of regression analysis, used to search the mathematical expression space in an effort to create a model that accurately fits to a given dataset. In symbolic regression combinations of mathematical *building blocks* are combined, including expressions, operators, and constants to form a model. These models, and the combinations of mathematical building blocks they are composed of, are modified by the GP system in order to find a solution.

Symbolic regression problems are the most commonly used benchmark problem used in Genetic Programming [38]. However, due to a proliferation of different problems an effective comparison of different approaches is often difficult to achieve.

3.1.2 Route-Finding

Route-finding, or path-finding problems, are a type of problem where a route must be found through a simulated environment, usually involving some form of obstacles, such as a maze [5].

Canonical route-finding problems used in GP include: **Traversal problems** such as *the lawnmower problem* [12], **Artificial Ant problems** such as the *Santa-Fe trail* [12].

3.1.2.1 Lawnmower Problem

The lawnmower problem, introduced by Koza [12], is a simple problem consisting of an $n \times m$ toroidal grid representing a lawn of grass and a controllable agent representing a lawnmower. The essence of the problem is to create a solution which is able to traverse the entire lawn. The solution is comprised of a set of movement instructions, corresponding to the route taken.

The lawnmower state consists of its current location in the grid and orientation (N, E, S or W). Two example instances of size 10×10 are shown in Figure 3.1

Adapted from: Griffiths, T.D., and Ekárt, A. [5]

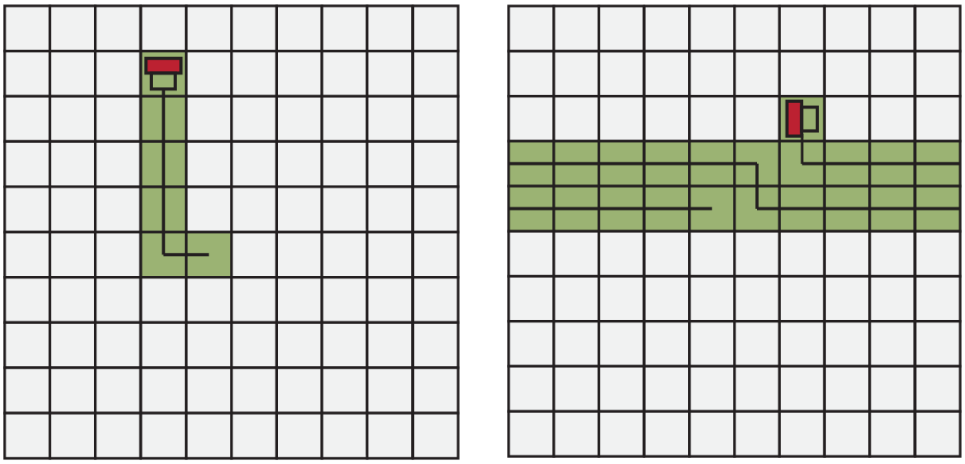


Figure 3.1: Example Instances of the Lawnmower Problem

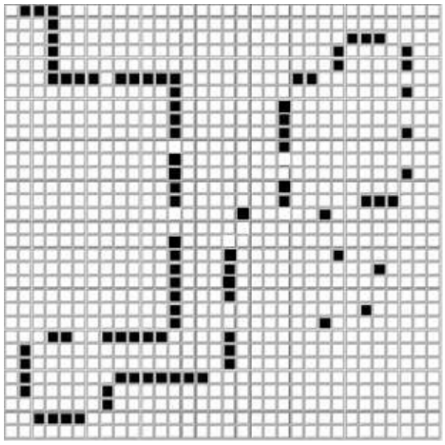


Figure 3.2: The Santa-Fe Trail Problem Instance

The shaded area shows the traversed lawn, while the line within indicates the actual trajectory taken.

3.1.2.2 Santa-Fe Trail Problem

The santa-fe problem, known also as the *artificial ant problem* [12] is a problem where an agent searches for food pellets, along a pre-determined route in a simulated environment. The agent has a finite amount of energy which is consumed as they move around the environment. Similarly to the Lawnmower problem, the solutions generated in the santa-fe trail are comprised of a set of movement instructions, corresponding to the route taken. The goal of the agent is to traverse the environment and collect the food pellets in the most efficient manner, utilising as little energy as possible.

3.1.3 Measuring the 'Best' Performance

The purpose of any benchmark problem is to provide a standardised test by which the performance of different approaches can be assessed and compared. This comparison is done in order to discern which approach is the *'best'*. The notion of

which approach is considered the best is one which is complex and often, at least partly, decided from a subjective point of view.

For many canonical benchmark problems, the means by which the best algorithmic approach is decided is done using the raw benchmark performance.
For example in symbolic regression problems, the algorithmic approach which produces the solution with the lowest error is typically considered the best.

3.2 The singular benchmark approach

Traditionally it was common for algorithmic approaches to be designed and tailored towards solving a specific problem, or specific set of pre-defined instances of a problem. This *singular* benchmark approach is effective during initial experimentation into the applicability and usability of an approach. It allows for benchmarks to be used to illustrate the performance and specific capabilities of a chosen approach.

3.2.1 Benchmarks as a Proof of Concept

When the field of Evolutionary Computation was in its infancy it was common for researchers and authors to use create their own problems in order to test their work, these were generally purposed as *proof of concept* problems [12]. As the field of research progressed and expanded, the de-facto benchmark problems continued to be used to test the relative performance of algorithmic approaches. The main drawback to using these problems is the fact that, in the majority of cases, the problems are trivially easy to solve. It is clear that using problems

that are trivially easy, does little to forward or improve the field of research [7]. Problems - especially ones that are used as benchmarks, should reflect real-world problems,be able to highlight features of an algorithm or provide domain insight, and be of a non-trivial difficulty [5].

3.2.1.1 The Simplicity Paradox

There are two primary reasons as for why many of the *proof of concept* problems are of trivial difficultly. Firstly, one approach to developing test problems was to take a problem from a real-world domain that the author was familiar with, and create a simplified representation of that problem in the digital domain.
It could be argued that for ease of construction and utilisation, these problems were often over-simplified and did not accurately represent the difficulty of the original problem.

Secondly, one must consider the reasoning behind the inception of the problems. There is little motivation to create a complex and difficult problem if it will only be used in order to prove the viability of an approach. To this end it may seem attractive to purposefully create a problem which is of trivial difficulty, in an attempt to further demonstrate a chosen approach.

3.3 The Benchmark Suite Approach

In order to combat the potential for over-fitting caused by singular benchmark analysis, modern benchmarking analysis approaches utilise a purpose made suite

of benchmark problems [7]. These suites contain several problems, usually with varying degrees of difficulty and dimensionality, allowing for a more complex analysis of multiple algorithm characteristics across a range of instances to be effectively carried out [38].

The push towards benchmark suites, and solutions whose performance is able to generalise across problems, was done in order to better emulate the real world. It is important under these circumstances that solutions and algorithm architectures are able to generalise and transfer performance in uncertain environments and ideally across several different but related problems domains.

3.3.1 Importance of Benchmarks

In order for a problem to be considered an effective benchmark candidate it must satisfy the majority of the aforementioned characteristics [38], combining them together to make an effective benchmark. However, many of the problems currently being used as de-facto benchmarks in GP only satisfy a small number of these characteristics [5, 7].

It is important to define a suite of benchmark problems, which collectively satisfy the entire range of desirable characteristics. The main benefit of having a suite of benchmark problems instead of using just a single problem is that it allows for different types of problems from various areas to be tested and the approaches compared. The range of problem domains also allows for the portability and scalability of a solution approach to be tested across the field of research.

3.3.2 Desirable Benchmark Characteristics

For many years there has been little agreement on what makes an effective benchmark in GP. More recently, following on from discussions at GECCO 2012 [38], a survey of the GP community by White et al. [7] outlined some of the important characteristics to be present across a suite of benchmark problems:

Tunable Difficulty One of the most important characteristics of an effective benchmark problem is tunable difficulty. A problem is said to be tunably difficult if there are methods by which the difficulty of instances can be changed and altered relative to each other. This provides scope for the benchmark to be used across a wide range of GP methods, while maintaining comparability between the results. The creation of instances of increasing difficulty is essential in order to push the boundaries of current research [38].

Precisely Defined A benchmark should be well-defined and documented, outlining the problem constraints and boundaries. It is common for a benchmark to be accompanied by a set of recommended resource constraints, such as an upper limit on the number of available evaluations or placing a time limitation on the program execution.

Accommodating to Implementors In order for a benchmark problem to be accepted by the research community, it must be accommodating to the practitioners who implement it, and straight forward to use. The benchmark problem must be self-contained and all its elements must be open-source and accessible, to ensure

universal access without the need for specific domain knowledge.

Representation Independent An effective benchmark should attempt, as far as it is reasonably practical, to be representation independent in terms of the programming language used and the programming style. As the field of GP expands and matures, we expect that the number of programming languages and representations being used will increase. Benchmarks should be flexible enough to allow adaptation between various languages and representations, while still being effective. Attempts should be made to ensure that the benchmark does not rely on any specific attributes from a language.

Easy to Interpret and Compare It is also important that the results generated by the benchmark are easy to interpret and can be compared without ambiguity. This clarity in understanding the results is vital, it allows for trends and relationships to be established between different sets of results, and reliable conclusions to be drawn on the data.

The intention was not to create an exhaustive list of every desirable feature of any benchmark suite, but a list of the key features that must be present in order for the set of benchmark problems to be utilised in an effective manner.

Chapter 4

The Tartarus Problem as a Benchmark

‘ ’

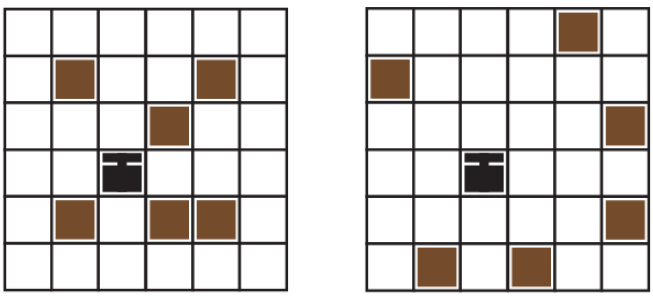
Chapter Contents

4.1 Introduction to the Tartarus Problem

The Tartarus problem (TP) is a grid-based optimisation problem defined by Teller [6] and introduced as a benchmark problem by Griffiths and Ekárt [5]. A TP instance comprises of an enclosed, non-toroidal grid environment of size $n \times n$, containing a predefined number of movable blocks B and a controllable agent A.

When a TP instance is initialised, the blocks and the agent are randomly placed within the central $n{-}2 \times n{-}2$ grid squares. Therefore, given a canonical TP instance of size $n = 6$, the blocks and the agent would be placed within the central 4×4 grid squares. An example initial state of size $n = 6$ with $B = 6$ blocks is shown in Figure 4.1(a).

Source: Griffiths, T.D., and Ekárt, A. [5]



(a) Example Initial State (b) Example Final State

Figure 4.1: Example Initial and Final Tartarus Instances.

Unlike other grid-based problems, such as the Lawnmower problem [12], the agent is initially unaware of its location and orientation within the environment. The agent receives input from eight sensors, allowing it to detect both blocks and the environment boundary in the surrounding eight grid-squares.

The agent is able to change its state by executing actions, chosen from the following three actions:

(1) *turn left*, **(2)** *turn right*, **(3)** *move forwards one square.*

The goal is to evolve a controller, allowing for the agent to locate the blocks and move the them to the edges of the environment within a set number of allowed actions m, an example goal state is shown in Figure 4.1(b).

At the end of a run, the environment is analysed and the agent is awarded a score, the fitness score, based on its progress toward achieving the goal.

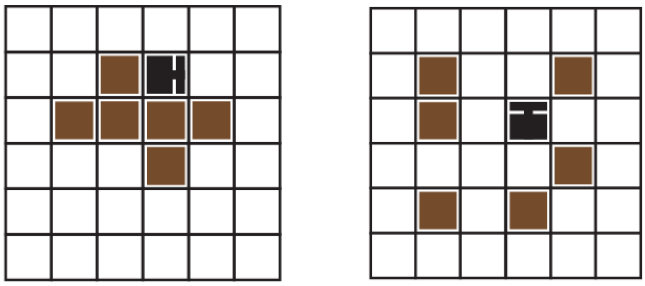
4.1.1 The Canonical Tartarus Instance

The canonical Tartarus instance consists of a grid of size $n = 6$, containing $B = 6$ blocks and one agent, example initial states are outlined in Figure 4.2.

As each tartarus instance is randomly generated, the placement of the blocks and agent can vary greatly between instances. In Figure 4.2(a) the blocks are placed in a clustered manner, with all of the blocks concentrated into one small area of the environment.

Conversely, in Figure 4.2(b) the blocks are placed in a more dispersed manner spread out throughout the grid. The placement of the blocks has an impact on the minimum number of moves required to complete the instance, for example the agent would require 32 moves to solve the clustered instance in Figure 4.2(a) and an agent would require only 27 moves to solve the dispersed instance in Figure 4.2(b), a reduction of more than 15%.

Source: Griffiths, T.D., and Ekárt, A. [5]

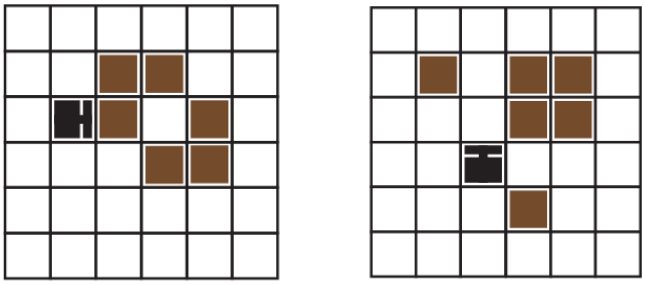


(a) Clustered Initial State (b) Dispersed Initial State

Figure 4.2: Example Clustered and Dispersed Tartarus Instances.

4.1.2 Generating Tartarus Instances

When generating Tartarus instances it is important to note that not all instances can be solved perfectly. It is likely that some instances will contain arrangements of blocks, which are impossible to move, making the instance partly or completely impossible to solve.



(a) Wilson Configuration (b) 4-Block Cluster

Figure 4.3: Partially Solvable Tartarus Instances

The two most common of these configurations are shown in Figure 4.3. The *Wilson* configuration shown in Figure 4.3(a), although it is possible to move some of the blocks, doing so is likely to create a cluster of four blocks, which cannot be moved. Similarly, Figure 4.3(b) shows an instance with an existing four block cluster which cannot be moved. Although the four block cluster is the

most frequent configuration preventing completion in the canonical 6×6 instance, it is also possible for all 6 blocks to be placed together in an immovable cluster.

4.1.2.1 Baseline Instance Values

In order to establish baseline values for use in the Tartarus problem, a set of Tartarus instances were created, with 1000 for each of the four chosen sizes, $n = \Big\{6, 7, 8, 16\Big\}$.

The number of instances generated and used for the experimentation was decided using the following equation [42]:

$$x \quad \geq \quad \frac{1}{-\ln\left(1-\epsilon\right)}\left(\ln\left(|H|\right) + \ln\left(\frac{1}{\delta}\right)\right) \qquad , \tag{4.1}$$

given a function f in a space of functions H, that x training examples suffice to ensure, with probability $1 - \delta$ that any hypothesis consistent with the data will not produce an error larger than ϵ.

Therefore, given a Tartarus function in a space of functions equal to 3^{80}, 1000 instance training examples would suffice to ensure that the data will, with a probability of 0.95, produce an error smaller than 0.1 .

Through experimentation on the generated Tartarus instances, a set of baseline instance values were established. A successful Tartarus solution must involve both finding the blocks in the environment and pushing the blocks to an edge, taking into account obstacles such as immovable block clusters.

Let us consider the step of pushing a block to an edge. The expected distance of

a block randomly initialised in a 6×6 grid to an edge is:

$$\text{Dist} = \frac{1}{(n-2)^2} \sum_{i=1}^{\frac{n-2}{2}} 4(n-1-2i)i = \frac{n(n-1)}{6(n-2)} \quad , \tag{4.2}$$

where $4(n-1-2i)$ is the number of positions on the inner grid of size $n-2$ that are at distance i from an edge of the grid of size n.

Therefore, if the agent encounters a block and *does not know the direction in which to move* to get to the closest edge, the expected distance to have to push a block at location (x, y) to any edge becomes:

$$\text{Dist} = \frac{x + (n-1-x) + y + (n-1-y)}{4} = \frac{n-1}{2} \quad . \tag{4.3}$$

For a grid of size $n = 6$, the expected number of grid squares an agent must travel is 2.5, increasing to 3.5 for a grid of size $n = 8$, and 7.5 for a grid of size $n = 16$. The fraction of the grid that the agent can be expected to travel to reach any edge will be:

$$\frac{\text{Dist}}{n} = \frac{n-1}{2n} \quad . \tag{4.4}$$

For a grid of size $n = 6$ the agent without global vision can expect to have to travel a proportion of 0.416 of n in order to move one block to the edge (not accounting for obstacles and travelling from the edge after successful move of one block to the next block). As n increases, this proportion approaches 0.5.

For a an instance of $n = 6$ it is easier to move a block than for an instance of size $n = 7$ (0.429) and substantially easier than an instance of size $n = 25$ (0.48). This does not account for the occurrence of impossible to move blocks in an instance. Based on Equation 4.4, we can conclude that the effort expected to move one block to the edge increases as the size of the grid n increases.

The number of blocks in the environment which must be moved, B, contributes to the overall difficulty of the problem instance, as more or fewer blocks must be moved within the finite number of allowed moves.

Baselines were determined through regression, following generation and evaluation of a set of 1000 instances, with a fixed number of moves for each of the four sizes, $m = \{80, 109, 142, 569\}$ respectively, and a varying number of blocks B. The determined baseline for establishing the number of blocks is as follows:

$$B_{baseline} = \left\lceil \left(n-1\right)^2 \cdot \left(\frac{1}{3} - \frac{n-1}{9n}\right) \right\rceil \quad . \tag{4.5}$$

The number of allowed moves m can be used to tune the difficulty of the Tartarus problem. It makes sense to link this resource limitation to the grid size, as the number of steps required for simply traversing the grid, or moving one block to an edge, depends on the grid size.

In order to establish the relationships between the problem parameters and pro-

duce reliable results, we determined that the following quadratic function would be suitable:

$$m_{baseline} = \left\lfloor \frac{20n^2}{9} \right\rfloor \quad . \tag{4.6}$$

It is recommended that the values calculated using the recommended baselines in Equation 4.5 and 4.6 be utilised as a reference when creating Tartarus instances. It is recommended that instances are created using a working range of ±25% from the baseline values, as outlined in Table 4.1. Values outside of this working range are likely to produce instances which are either trivially easy or potentially impossible to solve.

Table 4.1: Reference guide for generating Tartarus instances

n	Moves			Blocks		
	−25%	$m_{baseline}$	+25%	−25%	$B_{baseline}$	+25%
6	60	80	100	4	6	8
7	82	109	136	7	9	11
8	107	142	177	9	12	15
16	427	569	711	39	52	65
32	1707	2276	2845	163	217	271

4.1.3 Measuring Instance Difficulty

In order to compare agent performance across multiple different instances, a method of estimating the difficulty of instances is necessary. We proposed a generic method for estimating the relative difficulty based on the characteristics

of the instance, outlined in Equation 4.7.

$$D = \begin{cases} \dfrac{m_{baseline}}{2m} + \dfrac{B_{baseline}}{2B} + \dfrac{B_I}{B} & \textbf{if} \quad B_I < B \\ \textbf{impossible} & \textbf{if} \quad B_I = B \end{cases} \tag{4.7}$$

where $m_{baseline}$ is the baseline number of moves defined in Equation 4.6, m is the user set number of allowed moves, $B_{baseline}$ is the baseline number of blocks defined in Equation 4.5, B is the user set number of blocks, B_I is the number of impossible-to-move blocks.

Equation 4.7 allows for comparison of the relative difficulty between generated Tartarus instances. The difficulty is estimated using the differences between the suggested baseline values for the number of moves and blocks and the number chosen for the instance generation and the number of impossible-to-move blocks present in the instance.

4.1.3.1 Tuning Difficulty

The formula for estimating the difficulty of an instance, outlined in Equation 4.7, ensures a clear separation between instances that only contain impossible-to-move blocks and instances that have some movable blocks. In the case where there are movable blocks present in an instance, it accounts equally for the relative changes in the number of moves and number of blocks away from the suggested baseline values. Therefore, the added difficulty of having a number of blocks in an instance which are immovable is factored into the difficulty calculation.

The difficulty estimation has been designed to return an approximate value of 1 for instance using the suggested baseline values, although the inclusion of immovable blocks may vary this slightly, a value greater than 1 for more difficult instances and a value less than 1 for instances which are easier to solve.

Table 4.2 provides a range of estimated difficulty levels and the corresponding parameter values, from *very easy* to *very hard* for instances of size $n = \{6, 8, 16\}$. These examples reflect how reducing the number of allowed moves increases the difficulty of an instance in addition to reducing the number of blocks which would make an instance even more difficult, due to the fact that intuitively it takes longer to locate blocks which are spaced further apart.

Table 4.2: Example instances and their difficulty

		Very Easy	Easy	Baseline	Hard	Very Hard
6×6	m	96	88	80	66	55
	B	7	7	6	6	6
	D	0.85	0.88	1	1.11	1.23
8×8	m	170	156	142	116	104
	B	14	13	12	11	11
	D	0.85	0.92	1	1.11	1.22
16×16	m	704	662	569	486	422
	B	58	55	52	50	47
	D	0.85	0.90	1	1.11	1.23

4.1.4 State Evaluation

For the Tartarus problem a solution consists of a controller, which controls the actions of the agent in the environment. In order to test the efficacy of a solution, there needs to be a way to measure its outcome on an instance environment.

This is usually done by evaluating the end position (state) after executing the complete series of instructions. In fact, the same evaluation method can be used to evaluate any state, for different purposes:

- evaluate the initial state, in order to evaluate the problem instance
- evaluate an intermediate state, after a set period of time or number of moves, in order to measure progress
- evaluate the end state in order to evaluate solution quality.

4.1.4.1 Canonical State Evaluation

The originally established method of state evaluation only rewards the number of blocks, which have been completely moved to the edges of the environment. This binary success or fail approach works well for many benchmark problems where the absolute score achieved by a candidate solution is the only desired success measure.

However, for GP, rewarding part-way solutions is essential during evolution, so that better solutions can evolve. For example, the cluster of blocks in Figure 4.2(a) is very different from the dispersed blocks of Figure 4.2(b). Both these states would have the same evaluation score of zero.
As mentioned in Section 4.1.1, the dispersed blocks example in Figure 4.2(b) is visibly closer to an optimal end state, requiring only 27 to complete, compared to the cluster example in Figure 4.2(a), requiring 32 moves to complete.

The binary assignment of success or fail is too coarse to be practically useful. It misses important differences in performance between candidate solutions in the population. Solutions that move no blocks at all would be evaluated the same as solutions that make some progress, but fall short of actually moving a block to the edge of the grid (see Figure 5(b)). Solutions that have not actually moved blocks to the edges of the grid could still have made some progress by moving blocks *closer* to the edges and this should be recognised and rewarded in some manner.

According to the same rationale, initial states with the same number of blocks are not equivalent, as some require less moves than others to solve. Therefore the difficulty of the problem instance is not only dependent on the grid size, number of blocks and allowed number of moves, but also the initial distribution of blocks on the grid.

4.1.4.2 Improved State Evaluation

We propose a new evaluation method that rewards states according to how close they are to the desired final states, by including how close to the edge each block in the given state is:

$$State\_value = \mathrm{C}_1 \cdot \left(B - \frac{2}{n} \sum_{i=1}^{B} d_i - \mathrm{C}_2 \right) \tag{4.8}$$

where B is the total number of blocks, n is the size of the grid and d_i is the distance of block i from an edge in the given problem instance. C_1 and C_2 are

scaling and translation constants based on B and n in order to make the value range consistent with the current evaluation method. This allows for the direct comparison of results from instances of any size with the canonical instance of size n=6. For an instance of size n=6, with B=6 blocks $C_1 = 1.8$ and $C_2 = \frac{8}{3}$.

1000 random tartarus instances of size $n = 6$ were generated containing 6 blocks and evaluated using both the canonical evaluation and our proposed evaluation methods.

In Table 4.3 we present 30 randomly selected examples of this state evaluation comparison. As shown, the new method of state evaluation allows for a more fine-grained and accurate evaluation.

Table 4.3: Fitness data comparing evaluation methods

Individual	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Old Value	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
New Value	1.2	1.2	1.8	0.6	1.8	1.8	1.8	1.8	2.4	2.4	2.4	2.4	2.4	3.0	3.0
Individual	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Old Value	2.0	2.0	2.0	2.0	2.0	2.0	3.0	3.0	3.0	3.0	3.0	3.0	4.0	4.0	4.0
New Value	3.0	3.0	3.0	3.6	3.6	3.6	3.0	3.6	3.6	4.2	4.2	4.2	3.6	4.2	4.8

The distributions of the results for the canonical and proposed evaluation approaches are shown in Figures 4.4 and 4.5 respectively.

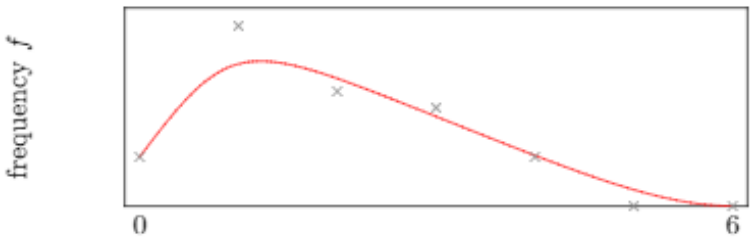


Figure 4.4: Current Evaluation Distribution

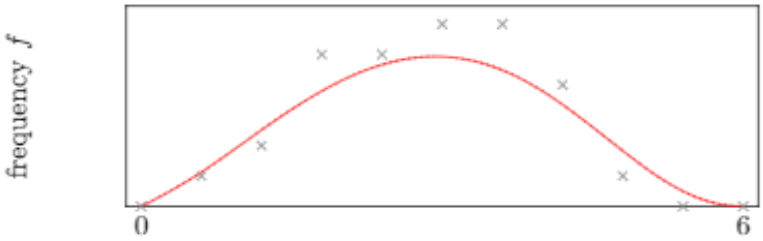


Figure 4.5: Proposed Evaluation Distribution

The current evaluation method rates all states with no blocks at the edges as equally poor, when in reality some can lead to end states that are just a few moves away from optimal end states. The proposed evaluation method rewards these. The distribution in Figure 4.5 better reflects the actual performance, at a much more fine grained level.

4.2 Satisfying the Desirable Benchmark Characteristics

The Proposed changes made to the Tartarus problem, outlined in Section 4.1.4.2, rewarding states according to how close they are to an optimal solution, lead to an increase in the granularity of the performance assessment culminating in the ability to tune the difficulty of a Tartarus instance.

The improved Tartarus problem satisfies the characteristics previously outlined in Section 3.3.2:

Tunable Difficulty The difficulty of the Tartarus problem can be altered by changing the parameters of the problem (grid size, number of blocks) and the restrictions placed upon the agent (number of allowed moves). This leads to a predictable shift in the complexity and difficulty of the generated instance.

Precisely Defined The Tartarus problem is a constrained problem with well defined boundaries for both the problem instance and the agent. The original problem definition outlined a maximum number of allowed moves, in this paper we suggest minimum and maximum constraints for allowed number of moves, number of blocks and the size of the instance, providing a comprehensive list of the operating constraints for the Tartarus problem.

Accommodating to Implementors Due to the fact that the Tartarus problem is a grid based problem it is simple to implement and use. There are no specialist skills or software tools that are required to create and use an implementation of the Tartarus problem.

Representation Independent As the implentation of the Tartarus problem is simple, it does not require any specific languages or software tools. The problem is representation independent and it should be possible to implement in any modern programming language and paradigm.

Easy to Interpret and Compare The improved method of state evaluation increases the ease with which Tartarus problem instances and solutions could be interpreted and compared, allowing for a more fine-grained evaluation of instances,

allowing for a more accurate comparison to be made.

Relevant There exists parallels that can be drawn between the Tartarus problem and real-world problems. For example, the recently announced Emergency Robots competition[1] (building upon the success of EuRathlon[2]), inspired by the 2011 Fukushima accident focuses on realistic emergency response scenarios. In these scenarios missing workers have to be found and critical hazards have to be identified in limited time.

Fast Given the improved method of state evaluation for the Tartarus problem outlined in the thesis, together with the simple nature of the implementation, tests can be carried out and meaningful comparisons can be made in a reasonable time frame. The fitness evaluation is fast, allowing for multiple executions per individual to be carried out.

4.3 Conclusion

The proposed changes made to the Tartarus problem, rewarding states according to how close they are to an optimal solution, lead to an increase in the granularity of the performance assessment and associated fitness reward. This allowed for a greater degree of analysis to be carried on Tartarus instances, leading to the establishment of a set of Tartarus baseline parameter values.

This analysis culminated in the creation of a reference guide for creating Tartarus

[1] http://eu-robotics.net/robotics_league

[2] http://www.eurathlon.eu An outdoor robotics challenge for land, sea and air

instances of sizes up to n=32. A range of parameter values and their associated difficulty level estimated were provided, from this it is possible to predict the relative difficulty of a given instance, with more difficult instances being harder to solve - either through a change in the number of allowed actions or the number of blocks present in an instance.

With regards to **RQ1**, the proposed modifications made to the Tartarus problem have identified it as a suitable benchmark problem, satisfying several of the desirable benchmark characteristics - specifically, the ability to tune the difficulty of Tartarus problem instances.

Chapter 5

Self-Adaptation in Genetic Programming

‘ ’

Chapter Contents

5.1 Self-Adaptation

In the field of evolutionary computation and specifically genetic programming, it is widely accepted that the on-the-fly modification [43] and adaptation of parameter values at runtime can lead to improvements in performance [44].

Self-adaption aims at biasing the distribution of individuals in the population towards more appropriate and effective areas of the search space at runtime [45]. This is achieved by means of setting and adjusting *control parameters* [8] - these can be expressed in terms of population size, recombination probability, mutation rate or the internal mechanisms of genetic operators, to name a few.

Rosenberg proposed [46] the adaptation of crossover probabilities, with Bagley [47] introducing the concept of incorporating the control parameters into the structural representation of the individual and Bäck proposing [48] the self-adaptation of mutation rates. This work was followed by Shaffer and Morishima in their work on self-adaptive *punctuated crossover* [49], concerning the adaptation of both the number and location of crossover points.

The desired outcome of self-adaptation is not only to find ways in which to improve the performance but to also to do it efficiently. It should be noted that this is further complicated due to the fact that the process of optimising the self-adaptation bias is itself a dynamic problem, since a set of control parameters which was deemed optimal at the start or during the run, may end up being unsuitable at a later stage in the evolutionary process.

It is for this reason that there is a need for a mechanism which favours a constant and continuous opportunity for modification during runtime [50].
We propose to conceptualise the modification of parameter values as two distinct processes, the first: *'when'* to modify and the second: *'how'* to modify.

5.1.1 When to modify

A common approach for deciding *when* to trigger the parameter modifications, whether they be deterministic [51] or probabilistic [52] in nature, is by use of a pre-determined schedule or fixed time interval; we refer to the parameter value modifications triggered by these methods as *episodic modifications*. The primary benefit of episodic methods is that they allow for a regular and predictable sequence of parameter modifications to be performed over time without the need for any further interaction.

However, the rigid nature of this approach presents drawbacks when utilised on dynamic multi-dimensional optimisation problems, such as the Tartarus Problem. During the execution of a run, it is likely that the environment of the problem may change, and the chosen time interval, which was deemed appropriate at the start of the run, may no longer be optimal and may need to be changed.

An alternative to *episodic modification* is to create a mechanism which provides a continual opportunity to modify parameter values at any time; we refer to this as *continuous modification*.
We propose the establishment and introduction of a self-adaptive crossover bias

method, as outlined in Section 5.2, allowing for the continual modification of individual crossover parameters at runtime.

5.1.2 How to modify

The process of deciding *'how'* a parameter value is to be modified is often more complex, this can be divided into two smaller, sequential sub-tasks:

- Deciding the mechanism by which the parameter values are modified,
- Calculating the magnitude of the parameter value modifications.

This division between the mechanism and the magnitude allows for the modifications and the impact of the modifications to be tuned and controlled separately at runtime. There exist several different approaches to deciding *'how'* the parameter values should be modified that are utilised in genetic programming, these can be classified as *deterministic*, *adaptive* or *self-adaptive*.

5.1.3 Parameter Modification Approaches

Here the three categories of parameter modification approaches utilised in genetic programming, *deterministic*, *adaptive* or *self-adaptive* are described.[1]

Deterministic Parameter Modification

The parameter value is modified on a *global* level according to a fixed, predetermined rule. The mechanism receives no feedback from, and is not influenced by, the current status of the search [53, 54].

[1]The descriptive terms 'Adaptive' and 'Self-Adaptive' are used in the broad general context of Evolutionary Computation. These terms have distinct meanings in fields such as Artifical Life; based on strict Ecological and Psychological definitions.

Adaptive Parameter Modification

The parameter value is modified on a *global* level according to a mechanism which receives input from, and is at least partly influenced by, the status of the search [55].

Self-Adaptive Parameter Modification

The parameter value is modified on an *individual* level, where the parameters are encoded into the genome of an individual in some form. The parameters undergo the same processes of mutation and recombination as the individuals. The modification of these parameter values is coupled with the status of the search [56].

In adaptive parameter modification (APM) the mechanism by which the parameter values are modified is defined in advance, leading to an explicit exogenous parameter modification. The performance of APM is only as good as the information that it receives from the environment, therefore care must be taken to ensure that the information received is applicable to the selected parameters.

Finally, in self-adaptive parameter management (SAPM) the way in which the parameter values are modified is entirely implicit. In this approach the mutation and recombination processes of the evolutionary cycle itself are used and exploited. The parameter values are embedded in the representation [57] leading to an implicit endogenous parameter modification. The performance of SAPM is closely linked to the choice of evolutionary operators, therefore effective operator choice is essential.

5.1.4 Taxonomy of Approaches

In order to classify and compare the different parameter modification approaches, taxonomies and classification schemes have been presented, with Angeline [58] considering the different types of adaptation to be divided between *absolute* and *empirical* update rules.

Absolute updates involve the sampling of an individual over several generations or sampling the population as a whole, using the outcome of which to make a decision based on a deterministic or fixed rule approach. By contrast, empirical updates control the value of the parameter values, often incorporated as part of the individuals structural representation and subject to the impact of genetic operators.

The classification proposed by Eiben et al. [44], builds upon and extends the concepts introduced by Angeline, dividing the approaches into three categories, *deterministic, adaptive* and *self-adaptive*, based on the level or scope of the adaptation, i.e. where the changes occur, either at individual or population level.

We propose an improved taxonomy, broadening the classification of Eiben et al. [44], allowing for comparison between the different parameter modification approaches to be made. Each approach is *affected by* a set of factors, both internal and external, which influence the overall effectiveness and performance. The self-adaptive approach leads to modifications to be made at the individual level, in contrast the adaptive and deterministic approaches both lead to modifications to be made at a global level.

Table 5.1: Taxonomy of Parameter Modification approaches. (× indicates a relationship.)

		Deterministic	APM	SAPM
Affected By	Explicitly-Defined Mechanisms	×	×	
	State of the Search		×	×
	Operator Selection			×
Modifies	Population Level Parameters	×	×	
	Individual Level Parameters			×
		absolute	*empirical*	

The deterministic parameter modification approach can be seen as analogous to the absolute updates rules proposed by Angeline [58], with APM and SAPM both being considered as forms of empirical updates rules, of varying degrees of complexity.

5.2 Self-Adaptive Crossover Operator

We postulate that allowing the GP system to trigger parameter modifications *'as and when they are required'* would reduce the number of ineffective adaptations being executed, increasing efficiency and allowing for convergence to an optimal solution.

We propose the implementation and introduction of a self-adaptive crossover operator, which is expected to create a more *continuous* parameter value modification process. The improved process would be more flexible, in comparison with the rigid, traditional *episodic* approaches outlined in subsection 5.1.1, leading to an increase in generated solution performance.

In order to test the efficacy of the proposed self-adaptive crossover operator, experiments were conducted using instances from both the Tartarus Problem (detailed in subsection 5.2.2) and the Santa-Fe Trail Problem (detailed in subsection 5.2.3). These problems were chosen due to the fact they share a number of similarities in terms of their structure: agent composure is similar with the agents in both problems are able to chose a finite number of movement operations from a set of three separate actions, in order to traverse the environment.

During the execution of a run, for each agent controller, the aggregate number of each of the three possible actions: *move forwards one square* (A_F), *turn left* (A_L) and *turn right* (A_R) alleles are counted. We refer to this underlying structure of the controller - akin to the genome, as the *composition*. It is important to note that this composition of the genome does not take into consideration the sequential order of the alleles, but only the aggregate number of each type of allele present.

We hypothesised that for both the Tartarus Problem and the Santa-Fe Trail Problem that there exist optimal compositions of agent actions, which, when used to seed future individuals, will likely lead to an increase in solution performance.

5.2.1 Crossover Bias Implementation

It was postulated that it would be possible to design a self-adaptive crossover operator bias, in an attempt to exploit the changes in expected fitness for different compositions of agent actions, residing in different areas of the composition space.

This allows for the introduction of bias in the generation of new individuals by favouring offspring with certain compositions. As it is the output of the chosen crossover operator that is affected, the process of generating new individuals, the proposed self-adaptations can be incorporated and utilised alongside any traditional crossover approach.

In order to implement the bias, the crossover operator was parameterised at the individual level. During initialisation each individual was assigned a random target A_F value T'_g, in the chosen range $A_F = \left[0.2m, 0.8m\right]$ allowing a wide spread of initial values excluding the excessively high and low values, from where the bias can adapt during evolution.

5.2.1.1 Updating Crossover Bias

The process of adapting the target value is divided into two stages. In the first stage, the *'how'* stage, the target value T'_g is updated at the end of generation g during the evaluation step, according to the performance of the individual in comparison to previous evaluations:

$$T'_g = \begin{cases} T_g & \textbf{if} \quad F_g > F_{g\text{-}1} \\ T_g + R_g & \textbf{if} \quad F_g \leq F_{g\text{-}1} \end{cases}, \tag{5.1}$$

where T_g is the current target value, F_g and $F_{g\text{-}1}$ are the current and previous fitness scores of the individual and R_g is a uniformly distributed random value

in the interval:

$$\left[-\frac{\mathrm{A_{F\,g}}}{\mathrm{T_g}} , \frac{\mathrm{A_{F\,g}}}{\mathrm{T_g}} \right] \quad ,$$

where $\mathrm{A_{F\,g}}$ is the current $\mathrm{A_F}$ value for the individual in generation g.

In the second stage, the '*when*' stage, the probability of triggering the self-adaptation and implementing the new target value $\mathrm{T'_g}$ into the crossover parameters of the individual is calculated:

$$P(\mathrm{T'_g}) = \frac{\mathrm{T_g}}{\mathrm{G} \cdot \mathrm{B} \cdot \mathrm{T'_g}} \quad , \tag{5.2}$$

where G is the number of generations since the last improvement in the fitness score of the individual, inclusive of the current generation, and B is the number of blocks present in the instance.

The probability $P(\mathrm{T'_g})$ is influenced by both the number of generations G since the actions of the individual led to an improvement in fitness score and the change between the target values $\mathrm{T_g}$ and $\mathrm{T'_g}$.

As G increases or the difference between $\mathrm{T_g}$ and $\mathrm{T'_g}$ increases, the chance that the self-adaptation will be triggered becomes greater. If the self-adaptation is triggered, at the start of the next generation, $\mathrm{T_{g+1}}$ will be initialised with the current value $\mathrm{T'_g}$.

5.2.2 Self-Adaptive Crossover: Tartarus Case Study

As the candidate solutions present in the Tartarus Problem are comprised of three primary components, their compositions can be illustrated more clearly using a ternary plot, in order to visualise the magnitude of the components present in the composition. A population of 1000 individuals were generated. These individuals corresponded to 697 unique genome compositions.

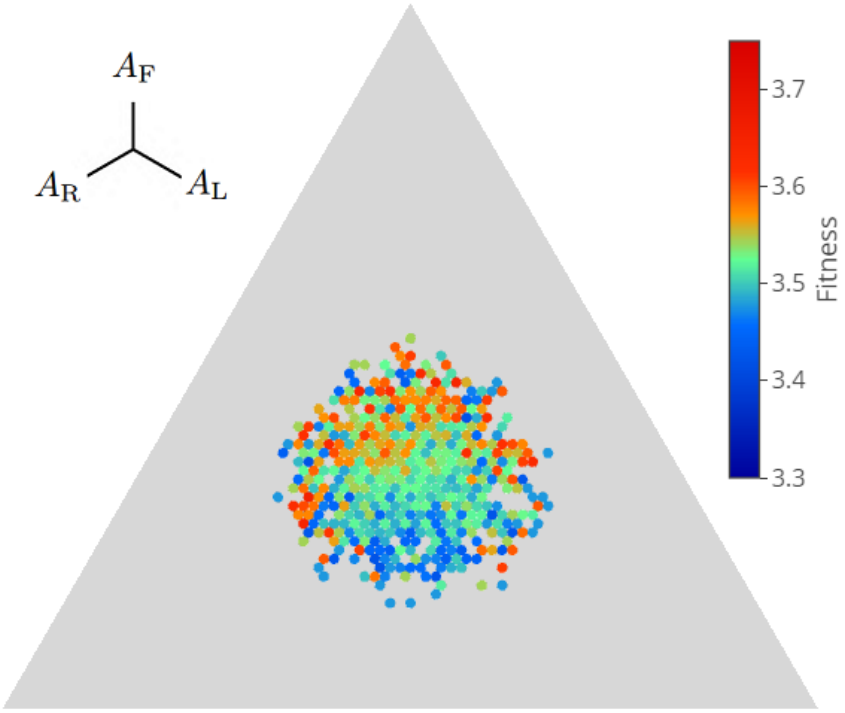


Figure 5.1: Central 80% of Compositions

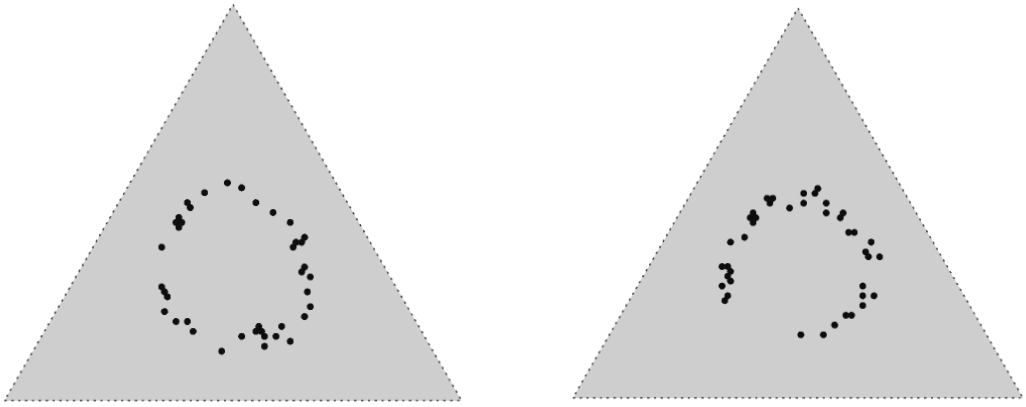
Utilising Equation 4.1 presented in Section 4.1.2.1, the population of generated individuals was executed across 1000 different Tartarus Problem instances of size $n = 6$, in order to get an indication of their performance. The resultant fitness scores averaged for each composition, as shown in Figure 5.1.

The placement of each individual datapoint in the ternary plot illustrates the magnitude of each of the three components present. Individuals with a higher level of A_F alleles, relative to the other alleles, will appear closer to the top point of the ternary diagram. Alternatively, individuals with a higher level of A_R and A_L alleles, relative to the other alleles, will appear closer to the leftmost and rightmost points of the ternary diagram respectively.

Upon inspection of the ternary plot in Figure 5.1, there is a clear divide in the average fitness between individuals who have an approximately equal composition, located in the central region of the ternary plot, and those individuals with an uneven composition, who lie on the periphery. 80% of the compositions fall within the central region; here the variation in average fitness score is low, with values in the range $\left[3.3\,,3.75\right]$.

However, for individuals who have uneven compositions, and fall outside of this central region; the variation in average fitness scores is high with values in the range $\left[2.6\,,4.6\right]$.

This is highlighted most clearly in Figure 5.2, showing the bottom 10% and top 10% of individual compositions in terms of their averaged fitness score. It can be seen that the top 10% and bottom 10% of compositions exist in two defined bands surrounding the central region, showing that moving forwards is necessary, but not sufficient on its own.



(a) Bottom 10% of Compositions (b) Top 10% of Compositions

Figure 5.2: Top and Bottom 10% of Compositions

It was hypothesised that increasing the number of *move forward* instructions in the genome, relative to the number of turn left and turn right instructions, would lead to an increase in fitness score.

This can be seen most notably in Figure 5.1; there is a defined change in fitness scores between the compositions in the upper section of the plot, with a higher proportion of A_F, and the compositions in the lower section of the plot, with lower proportion of A_F relative to the other alleles present in the composition.

This is expected behaviour: it is intuitive that compositions containing a high proportion of *turn left* and *turn right* instructions would simply spin around and not move very far from the initial grid location, therefore having a lower score. In a similar manner, compositions containing a lower, but approximately equal, number of *turn left* and *turn right* instructions, the impact of which would effectively be cancelled out, result in a lower score.

In order to test whether the self-adaptive crossover operator was effective at proving the hypothesis, a second population of 1000 individuals were generated. However, in contrast to previous experiments, the instances used were of size $n = 8$, a more complex and harder problem than the canonical Tartarus instance of size $n = 6$. Therefore each individual had a genome containing a random mixture of $m = 142$ alleles, longer than the canonical $m = 80$ alleles.

These individuals were tested on 100 different instances of size $n = 8$. The target A_F values T chosen by the individual at each generation g were averaged, as shown in Figure 5.3.

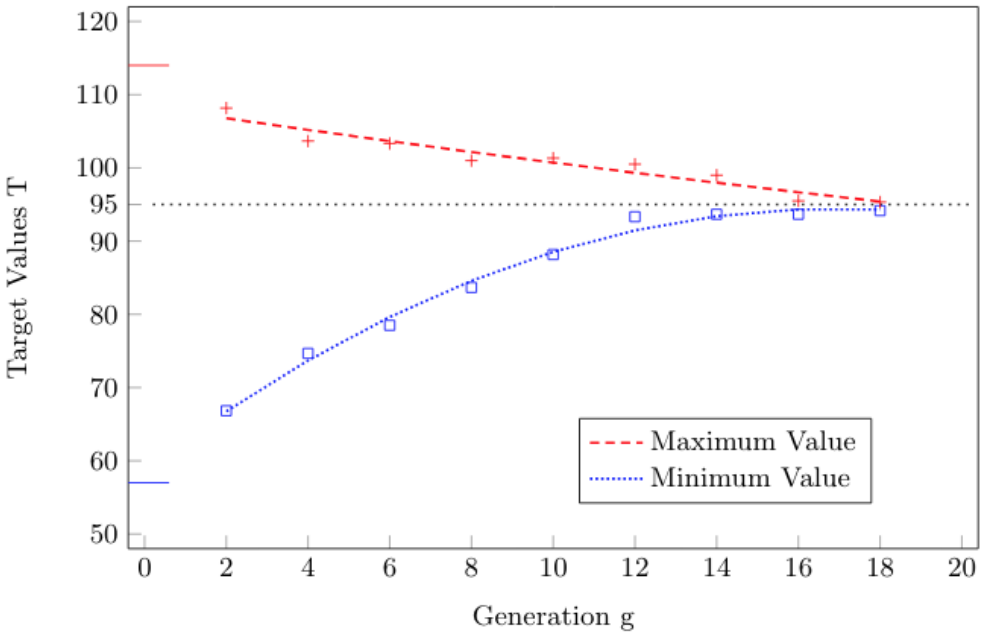


Figure 5.3: Convergence of Target A_F Value T within the Population

Over time, the target values chosen by the individuals within the population stabilise and converge to a small range of values. Figure 5.3 also shows the maximum and minimum T values within the population, over generations, until they converge.

It can be seen that by generation 18 the target values of all the individuals within the population have converged to approximately $A_F = 95$, for an instance of size $n = 8$. This indicates that allowing for the self-adaptation of the target value T leads to the creation of a crossover operator favouring individuals with compositions with close to optimal A_F values.

Comparing the results to previous experiments we can conclude that an A_F value close to the optimal value has been found in Figure 5.3.

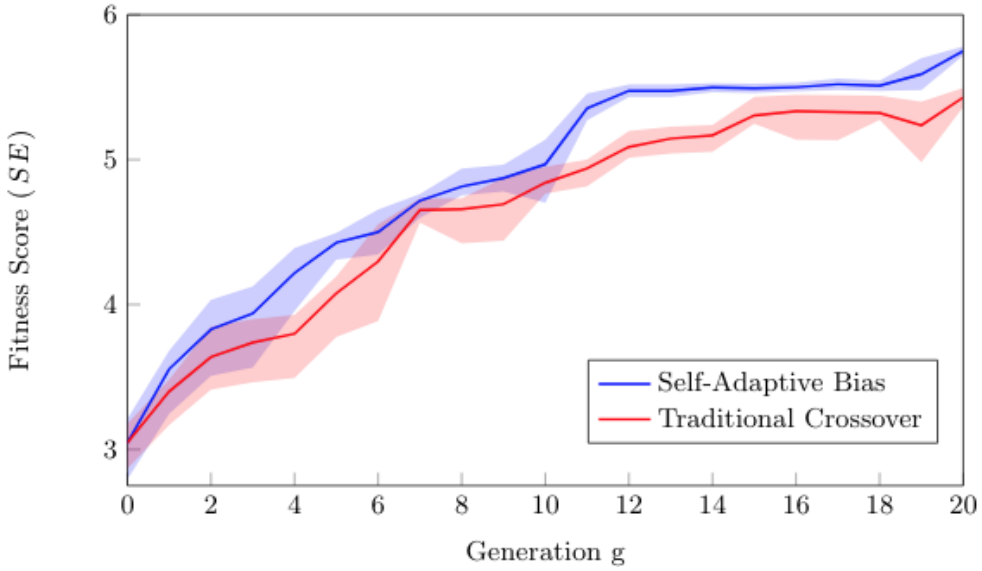


Figure 5.4: Comparison between Self-Adaptive Bias and Traditional Crossover

In Figure 5.4 the performance of the self-adaptive crossover bias, averaged over 20 different Tartarus instances of size $n = 8$ with $b = 12$ blocks, is plotted against the performance using standard canonical crossover. The range in fitness values present in the population at each generation is shown by the shaded areas, with the average score shown as a solid line.

The occurrences of self-adaptations being triggered within a population plotted against the changes in maximum fitness score, on a generation by generation basis is shown in Figure 5.5.

The plot shows that there is a strong correlation between the occurrence of self-adaptations within the population and an increase in the maximum fitness score achieved.

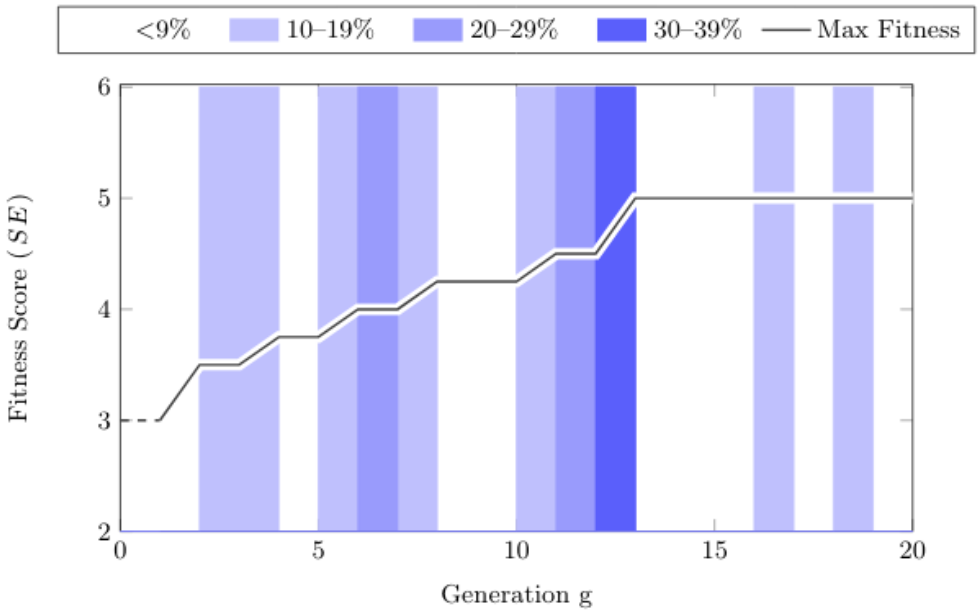


Figure 5.5: Occurrences of Self-Adaptation and the Maximum Fitness Score.

Between generation 11 and generation 12, 32% of the individuals in the population triggered self-adaptations of their target A_F value T_n. This coincided with an increase of 0.5 in the maximum fitness score present in the population, bringing it from 4.5 to 5.0. This is a substantial increase in the maximum fitness score of the population, a likely consequence of the self-adaptations carried out by the individuals.

5.2.3 Self-Adaptive Crossover: Santa-Fe Case Study

Similarly to the the candidate solutions in the Tartarus Problem in Section 5.2.2, the candidate solutions present in the Santa-Fe Trail Problem are comprised of three primary components, and their compositions will be illustrated using a ternary plot. A population of 1000 individuals were generated.

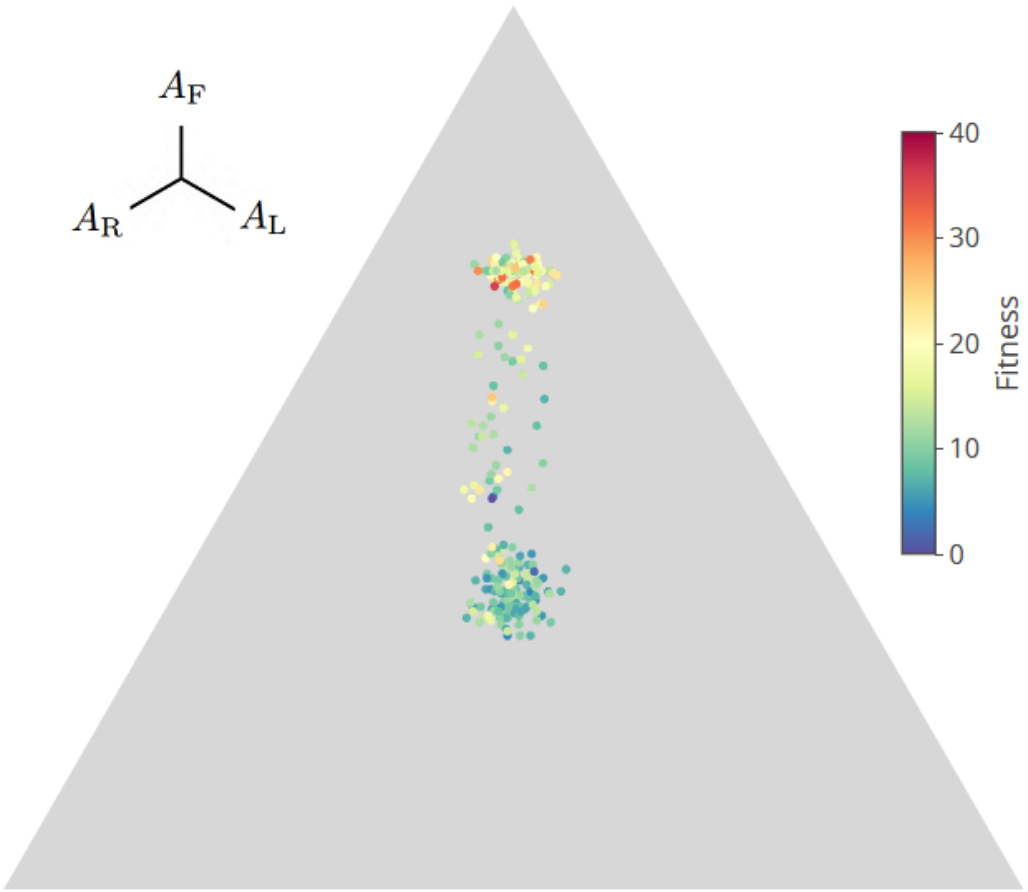


Figure 5.6: Ternary plot of Santa-Fe Trail Execution

The generated individuals were executed on Santa-Fe Trail Problem instances, in order to get an indication of their performance. The resultant fitness scores averaged for each composition as shown below in Figure 5.6. It is clear from the plot that two main clusters of compositions emerged. Following on from the

conclusions of the Tartarus Problem experiments in Section 5.2.2 the individuals in the Santa-Fe Trail Problem appear to exhibit an increase in performance as the number of A_F alleles are increased relative to the other alleles in their compositions.

However, unlike in the Tartarus Problem results where a single cluster of individuals was present, in the Santa-Fe Trail Problem the change in expected performance appears to be amplified and two clusters of individuals have formed. This has made the performance differences between the two groups of individuals clear to see, the individuals who have increased proportion of A_F alleles, in the topmost cluster, have a higher level of performance in comparison to the individuals with a lower relative proportion of A_F alleles, in the central cluster.

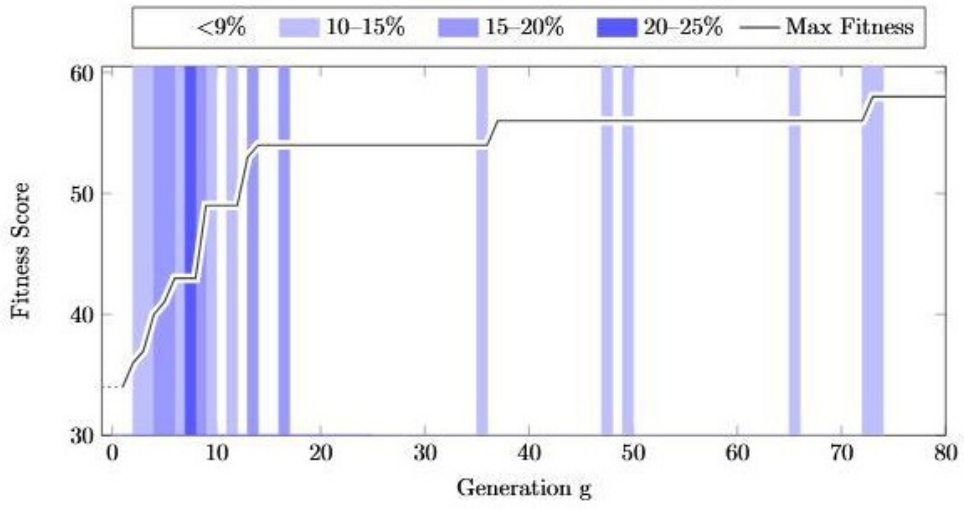


Figure 5.7: Occurrences of Self-Adaptation and the Maximum Fitness Score.

The individuals in the Santa-Fe Trail population are exposed to the same self-adaptive crossover operator as the individuals in the Tartarus Problem population. The occurrences of self-adaptations are triggered within the population

plotted against the changes in maximum fitness score, on a generation by generation basis is shown in Figure 5.7. The plot shows a correlation between the occurrence of self-adaptations within the population and an increase in the maximum fitness score achieved.

5.2.4 Results

The relationship between the proportion of individuals in a GP population undergoing self-adaptation and the subsequent increase in performance, averaged across both the Tartarus and Santa-Fe Trail problems, is shown in Figure 5.8. The increase in solution performance is averaged across a 3-generation window from the measured change in individuals undergoing self-adaptation, allowing for the on-going impact of the self-adaptation to assessed.

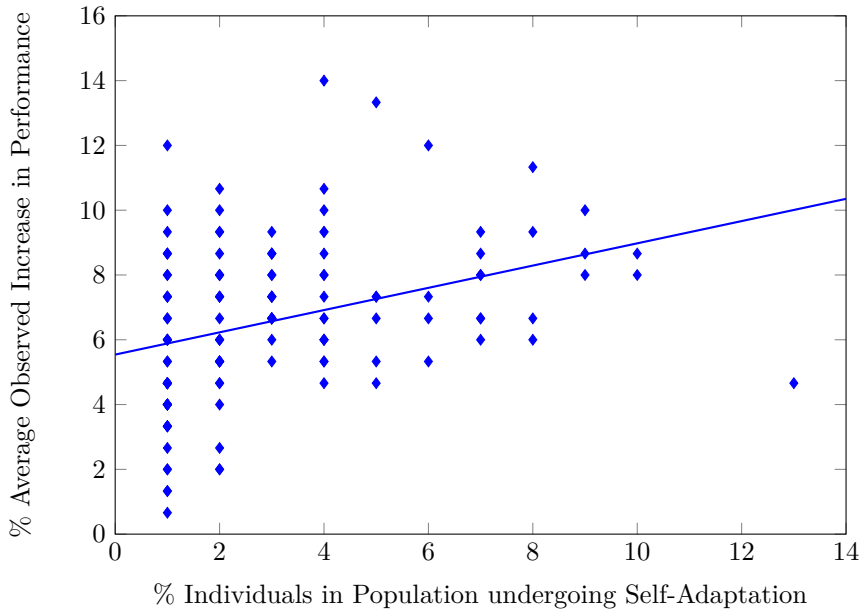


Figure 5.8: Relationship Between Changes in Self-Adaptation and Solution Performance

There exists a positive correlation in Figure 5.8, between the change in the number of individuals undergoing self-adaptation and the change in observed solution

performance over the 3-generation window. This supports the supposition that the introduction of the self-adaptive crossover operator in a GP system, leads to a measurable increase in solution performance throughout a run.

5.3 Conclusion

The proposed self-adaptive crossover operator provides a continuous opportunity for modifications to be made at runtime, rather than sticking to the more rigid, deterministic schedule. This increase in flexibility lead to an increase in effective modifications being made. The proposed operator was able to satisfy the desired outcome of self-adaptive systems outlined in Section 5.1, finding a way to improve performance whilst maintaining the overall efficiency of a system.

In relation to **RQ2**, the introduction of the proposed self-adaptive crossover operator into the GP system lead to an approximate overall average increase of 15% for the Tartarus problem, and 10% for the Santa-Fe Trail problem.

Chapter 6

Aspects of Robustness in Genetic Programming

‘ In the field of observation, fortune favours only the prepared mind. ’

Louis Pasteur

Chapter Contents

6.1 Introduction

Is it possible to define a relationship between the genotype, the structure of an individual, and the phenotype, the behaviour of an individual?

This question has dogged the progression of evolutionary theory since the early twentieth century [59]. Similar postulations continue to be relevant in evolutionary computation and genetic programming.

Much of the features and population characteristics present in the natural world can be replicated and observed in artificial populations in the digital domain. Specifically, the concepts of *robustness* [60, 61] and *diversity* [1] have been explored and analysed in artificial populations [62].

In this chapter we introduce and implement a fitness bias, based on the cognitive impact [63] of the Dunning–Kruger effect [64, 65], exploring the resultant impact on the level of population diversity and robustness in a GP system.

6.2 Modern Synthesis

The answer proposed by the architects of Modern Synthesis [66], the generally accepted biological model of evolution,[1] is that the genotypic make-up of individuals determines the resultant phenotypic actions; a common metaphor of which is the oft-cited '*genetic blueprint*'.

[1]Modern Synthesis is a mathematical framework proposed by *Julian Huxley*, combining and reconciling *Charles Darwin's* theory of evolution and *Gregor Mendel's* work on heredity. This combination of *Population Genetics* and *Mendelian Genetics* addressed the relationships between the broad-scale *Macroevolution* shown in the fossil record and the small-scale *Microevolution* of populations, exhibited by living organisms.

Since the discovery of genes as the sole units of heredity, there has been a logical push to view the genotype of an individual as the determinant of form. This is supporting the view that the genotype ultimately controls the developmental processes, which in turn, generate the form, predicating the actions of the individual. However, this depiction of the genotype and the development processes as stages of a hierarchical model is incorrect [67]. The genotype does not specify itself the developmental processes, nor the form of an individual; the genotype should be viewed as one of several causal factors that are jointly determinant of the phenotype of an individual [68].

Alberch introduced a new conceptual genotype–phenotype relationship, differing from the previous 'genetic blueprint' model [68]. He suggested that the relationship between the genotype and phenotype of an individual could be represented as a mapping function $f(\mathrm{G} \rightarrow \mathrm{P})$, defined by a given parameter space.

6.2.1 Genotype – Phenotype Map

The proposed mapping between the genotype and phenotype is not a simple one-to-one relationship. In the natural world the links between genotypes and phenotypes are complex and non-linear, the same phenotype can be expressed from several different and distinct genotypes.

The same relationship between phenotypes and genotypes is present in genetic programming in the digital domain. An example of this is *symbolic regression* where large solutions and smaller, simplified solutions can share the same pheno-

type despite having different, albeit equivalent, genotypes [69]. The many-to-one relationship indicates that the genotype space G is larger and more numerous than the phenotype space P, as illustrated in Figure 6.1.

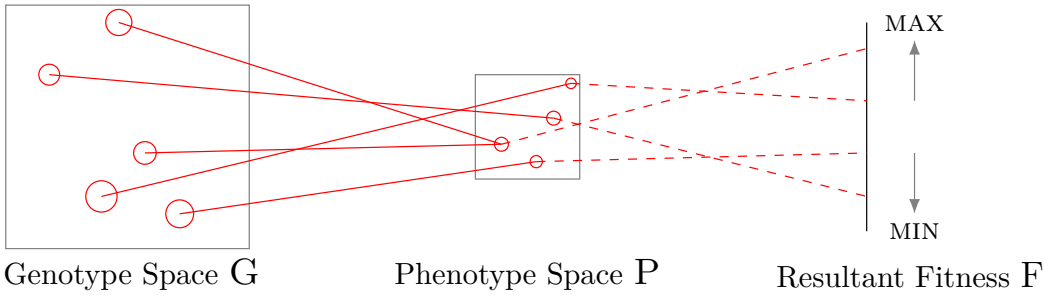


Figure 6.1: Example (G → P) Map:
Showing the relationship between a given set of genotypes G′, the associated set of phenotypes P′ and the resultant fitness evaluation F.

According to Pigliucci [69], the genotype should be treated as one of several causal factors that are jointly determinant of the phenotype of an individual. This is due to the fact that developmental events are both affected by, and also have an affect on, genetic expression. Therefore the parameters of the mapping function $f(\mathrm{G} \rightarrow \mathrm{P})$ are developmental in nature, with their values being determined by both environmental factors and genotypic expression.

6.3 Robustness

Robustness is referred to as a characteristic of a candidate solution whose performance is not diminished despite perturbations in environmental parameters or constraints [70]. A solution that does not lose its utility or performance quality under these changes is said to be robust [71]. Robustness can generally be classified into two groups:

Phenotypic Robustness – the number of unique genotypes that map to a given phenotype, and,

Genotypic Robustness – the ability of a genotype to produce the same phenotype under single-point mutation.

There exists a trade-off between the quality and the robustness of a generated solution. As a consequence, optimally performing solutions often have a sub-optimal level of robustness, such solutions are likely not to be resilient to small environmental or parameter perturbations [72].

Robustness is considered to be a positive characteristic of evolutionary computation solutions [73], due to the fact they are often able to maintain a satisfactory level of performance under dynamic and uncertain circumstances. It is possible to illustrate the relationship between genotypes and phenotypes.

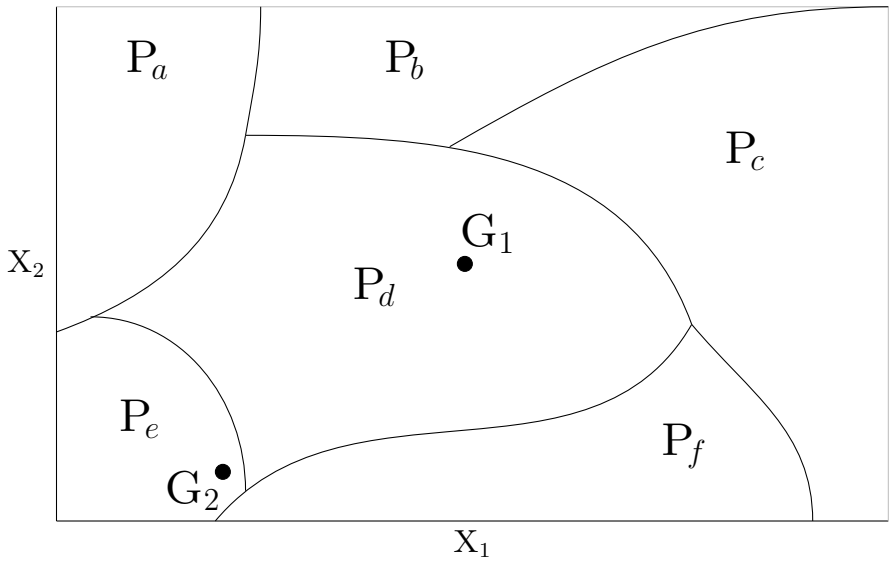


Figure 6.2: Example Two-Dimensional Geno-Pheno Space for X_1 and X_2.

Figure 6.2 shows a hypothetical geno-pheno space,[2] illustrating as examples, six phenotypes P_{a-f} and two genotypes G_{1-2}, as determined by the developmental interactions of the two parameters X_1 and X_2.

The figure supports the assertion that many different combinations of parameter values and different individual genotypic configurations can lead to the same resultant phenotypic outcome. Each phenotype occupies a separate area of the parameter space and these areas are separated by *transformational boundaries*, also known as *bifurcation boundaries*[3] [74]. Figure 6.2 can be used to illustrate the characteristics of phenotypic and genotypic robustness [67], as detailed.

6.3.1 Phenotypic Robustness

The robustness of a phenotype is defined relative to the size of its genotype network, as the number of unique genotypes present in the population that map to the given phenotype [75]. Intuitively, a phenotype which has a large genotype network is considered to be more robust than a phenotype with a smaller genotype network.

The size of the genotype network can be conceptualised as analogous to the size of the area of the geno-pheno space[2] occupied by the individual phenotype, indicating that a larger range of parameter values could be used to achieve the same phenotype.

[2] Although the parameter space shown in Figure 6.2 is 2-dimensional, the concept can be utilised in n dimensional space, outlining the interactions between n parameter values.

[3] In the context of *Dynamical System Theory*, transformational boundaries are often referred to as bifurcation boundaries, with the terms being used interchangeably.

The larger this area, the larger the number of unique genotypes that map to the phenotype, making the phenotype more robust and resilient to perturbations in parameter values. In Figure 6.2 the phenotypes P_c and P_d occupy a much larger area of the geno-pheno space than the phenotypes P_a and P_e, and would therefore be considered to be more robust.

6.3.2 Genotypic Robustness

Canonical GP genotypes have been compared to each other in terms of their n-neighbour relationships; genotypes which have *one* chromosomal difference are said to have a *1-neighbour* relationship, genotypes with *two* chromosonal differences are said to be have a *2-neighbour* relationship, and so forth [76, 69]. Genotypes with a 1-neighbour relationship can be conceptualised as as being adjacent to each other in the wider geno-pheno space.

A genotype is considered robust if it is able to map to the same phenotype under the effects of single-point mutation [75], which can be viewed in terms of its 1-neighbour relationships. It is considered more likely that for genotype G_1 in Figure 6.2, the genotypes with a 1-neighbour or 2-neighbour relationships still map to the same phenotype, P_d, these are known as *neutral neighbours* [76].

However, for the genotype G_2 in Figure 6.2, which lies closer to the transformational boundaries between the phenotypes, P_e, P_d and P_f, it is more likely that the genotypes with a 1-neighbour or 2-neighbour relationship may map to an entirely different phenotype, these are known as *non-neutral* neighbours.

The robustness of a genotype can be measured by comparing the number of neutral neighbours in relation to the number of non-neutral neighbours [75]. This can be conceptualised as being analogous to the distance between the genotype and the transformational boundaries present in the geno-pheno space in Figure 6.2. The larger the distance between the individual genotype and the transformational boundaries, the more robust the genotype is considered to be.

6.3.3 Transformational Diagrams

In order to simplify the relationships between phenotypes, we can derive the possible pathways of transformation from a specific parameter space. From these pathways we can construct a transformational diagram, where the nodes represent the phenotypes and the links between them correspond to neighbour relationships, as shown in Figure 6.3.

The arrows between the phenotypes in Figure 6.3 represent a shared *transitional boundary*, indicating that an individual with a genotype which lies near the boundary could potentially transition between the two given phenotypes.

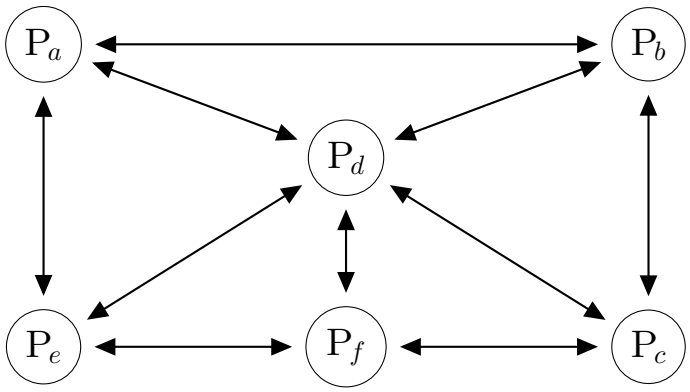


Figure 6.3: Transformational Diagram corresponding to the 'Geno-Pheno Space' shown in Figure 6.2.

In order to further enhance the transformational diagram, we are able to construct specific transformational diagrams for individual phenotypes, allowing us to differentiate between probable and improbable transitions, the transitions for phenotype P_f are shown in Figure 6.4.

,

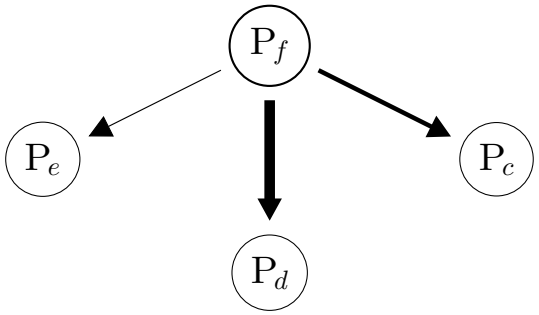


Figure 6.4: Transformational Diagram for the P_B Pheontype shown in the 'Geno-Pheno Space' in Figure 6.2.

This analysis does not take into account the relative viability of the adaptations. The probability of a particular transformation is defined to be proportional to the length of the boundary between the phenotypic domains shown in Figure 6.2. More probable transformations are shown as thicker lines and less probable transformation are shown as thinner lines.

6.4 Simulated Dunning-Kruger Effect

The Dunning-Kruger effect (DK) is a form of cognitive bias observed in populations, first described by psychologists Dunning and Kruger in 1999 [64]. Dunning and Kruger describe how individuals with a low level of ability mistakenly overestimate their performance and conversely, individuals with a high level of ability will often mistakenly under-estimate their performance. An example of this cognitive bias is shown in Figure 6.5.

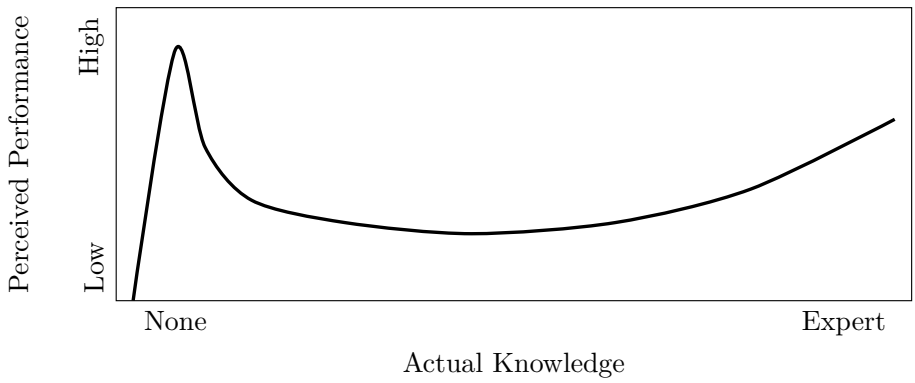


Figure 6.5: Illustration of the Dunning-Kruger Effect: Showing the relationship between the *Actual Knowledge* of an individual and their *Perceived Performance*.

The cognitive bias present in the Dunning-Kruger effect is clearly illustrated in Figure 6.6, highlighting the difference between the actual and perceived test scores for an experiment measuring humour recognition [64]. The area shaded in blue with vertical lines is the area where the perceived score exceeded the actual score, the area shaded red with dots is where the the perceived score is less than the actual score achieved by the individual.

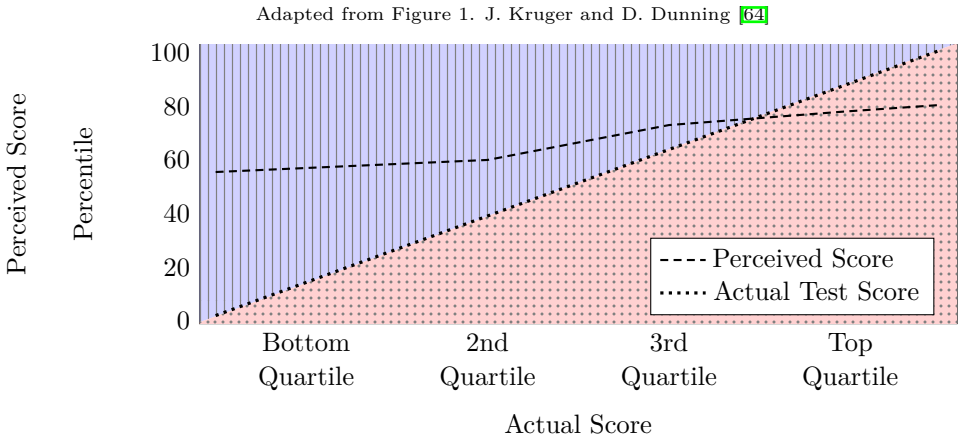


Figure 6.6: Comparison Between *Actual* and *Perceived* Test Scores

Although the reasoning behind the cognitive bias is complex, it is postulated that the main influencing factor is the fact that individuals with a low level of ability do not posses the fundamental knowledge or experience required in order to accurately and objectively assess their level of performance on a given task. These individuals are unable to assess the competence of others, and more specifically, their *own* level of competence.

Similarly, individuals who posses a high level of ability are likely to have the fundamental knowledge and experience required to be able to recognise the true scale and complexities of a given task. This often leads to the individuals inaccurately underestimating their performance, illustrated by a variation of the Socratic paradox, 'I know enough about x to know that I know nothing.'

6.5 Diversity

In genetic programming, the term *population diversity* is used to describe the level of similarity between individuals in a population. This can be measured by a variety of methods, which similarly to the robustness measures in Section 6.3, can be broadly classified into two groups:

Phenotypic Diversity – diversity in the effects as a result of the execution of individuals, and,

Genotypic Diversity – diversity in the structure of individuals.

Any measure of diversity used on a collection of objects must be able to reflect and quantify how similar, or dissimilar, the objects are in relation to each other.

Counting the number of unique individuals present in a population - known as the *variety* of a population [1], is a trivial task.

However, quantifying exactly *how* different the individuals in a population are - the *diversity* of a population, is a more complex task [77]. It should be noted however, if the variety of a given population is low then the diversity measure is likely to also be low [78].

6.5.1 Phenotypic Diversity

Phenotypic diversity methods measure and compare the functional differences between individuals, concentrating on the observed behaviours of individuals during execution [79]. One common approach for estimating the phenotypic diversity of a population is to calculate the *fitness spread* [80]: that is, the range of fitness values obtained upon evaluating the individuals within the population, during a given generation.

Many commonly used phenotypic diversity measures, such as the fitness spread approach, use the fitness evaluation values of individuals as an approximation of individual behaviours. However, for many problems, including those that have a discrete set of possible fitness values, there may be several substantially different individuals that have the same, or similar fitness value. These individuals can have a wide range of differing behavioural attributes but the same fitness value, leading to a lower, and seemingly inaccurate, measurement of the phenotypic diversity [81].

6.5.2 Genotypic Diversity

Genotypic diversity methods often utilise some form of distance measurement [82] or other quantitative measures [1], to compare the differences between individuals in the population. These methods are concerned with comparing the structural elements of individuals: that is, their shape, their depth, their overall size and the functional operations and terminal values used at each node.

Traditionally, one common approach to measuring the genotypic diversity of a population is to calculate the *edit distance* - the shortest sequence of editing operations required to transform between two individuals [83, 84], for all individuals in the population.

As outlined in Section 6.2.1, it is possible to have several different genotypes that produce the same phenotype, this *many-to-one* relationship between several genotypes and a single phenotype allows a population to have a substantial level of genotypic diversity but express a negligible amount of phenotypic diversity. Therefore, making efforts to maintain a higher level of genotypic diversity, such as niching [85], are not guaranteed to translate to higher level of phenotypic diversity.

It is expected that there is a time delay between a change in genotypic diversity and any resultant change in the level of phenotypic diversity present, reinforcing the need for mechanisms to operate on a continuous basis, such as described by the proposed self-adaptive crossover operator in Section 5.2.

We propose that the introduction of the simulated Dunning-Kruger bias into the fitness distribution of a population will enable the GP system to maintain a higher level of population diversity over time, leading to an increase in the robustness of the GP system.

6.5.3 Implementing Simulated Dunning-Kruger Bias

The DK bias will be achieved by means of modifying the fitness scores of individuals based on their performance relative to the rest of the population. The individuals will have their *actual* true fitness scores modified, returning their *reported* fitness score.

The lower performing individuals will have their *reported* fitness scores artificially increased and the higher performing individuals will have their *reported* fitness scores artificially reduced.

The *reported* fitness score of each individual i is calculated at the end of the generation, prior to the execution of the selection and recombination operators, using the following function:

$$\mathrm{DK}_i \quad = \quad F_i \times 50 \quad - \quad 0.75p \times \frac{F_{max} - F_{min}}{2} \quad ,$$

where DK_i is the biased, *reported* fitness score and F_i is the *actual* fitness score of individual i. The constants are a linear approximation, corresponding to the perceived test score in Figure 6.6, p is the percentile *actual* fitness rank of the individual, F_{min} and F_{max} are the min and max fitness scores found in the *actual* fitness distribution of the population.

The proposed approach of modifying and biasing the fitness distribution is similar to that of *fitness sharing* [86]. However, in contrast to fitness sharing, where the fitness of individuals is modified in relation to a distance metric or neighbourhood-based measurement - the fitness value of each individual in DK is modified purely in relation to their performance in comparison with the overall performance of the population.

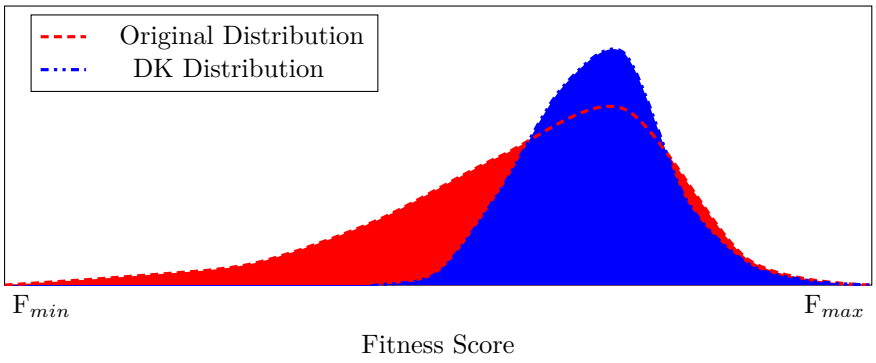


Figure 6.7: Comparison of the Original and DK Fitness Distributions

In Figure 6.7 the *actual* fitness and the *reported* DK fitness bias distributions, averaged across 100 tartarus instances of size n=8 are shown. It can be seen that the *actual* fitness distribution is skewed, with a significant minority of individuals whose performance lies around the median value, but who are unlikely to be selected for recombination; contributing, in part, to the long-term loss of overall population diversity. At the same time, the *reported* fitness distribution is closer to a more symmetrical distribution.

We hypothesise that the introduction of the DK bias into the fitness distribution will lead to a reduction in the evolutionary pressure present in the population

in a controllable and predictable manner, leading to a higher long-term level of diversity.

6.5.4 Results

In order to assess the efficacy of the DK bias at increasing the robustness of candidate solutions in a GP population, a selection of 100 tartarus instances of size $n = 8$ were generated.

Experiments were conducted where the instance was updated during execution at regular time intervals $C = \{10, 15, 20, 50\}$. This change in environment and instance setup was used to assess the robustness of the candidate solutions to changes, both small and large, during execution.

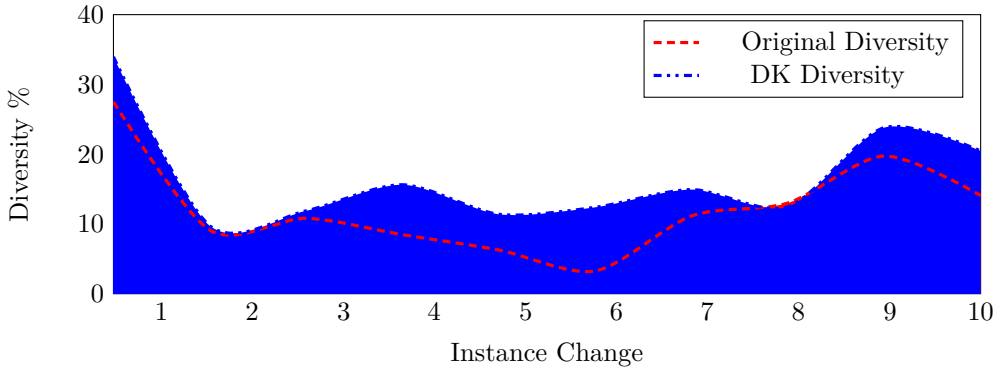


Figure 6.8: Average Level of Phenotypic Diversity Throughout Execution for a typical run of tartarus instance of size n=8, C=10 over 100 generations.

Figure 6.8 shows the average level of phenotypic diversity present in the population for both the DK and canonical GP systems during a typical run. This was calculated using a variation of the *fitness spread* technique [80]; the percentage of individuals with a unique fitness score. This measure utilises the *actual* fitness values present in the population, opposed to the *reported* fitness values modified

by the DK bias. The results indicate that the use of the *reported* fitness values leads to an increase in the overall level of diversity present in the GP population.

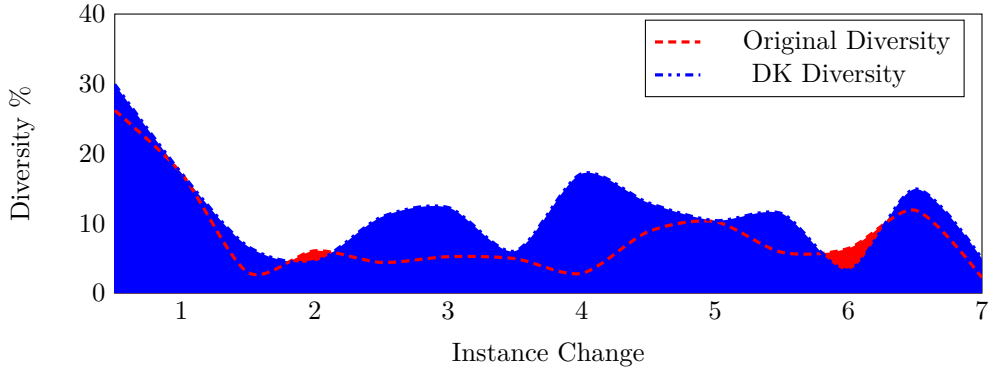


Figure 6.9: Average Level of Phenotypic Diversity Throughout Execution for a typical run of tartarus instance of size n=8, C=15 over 105 generations.

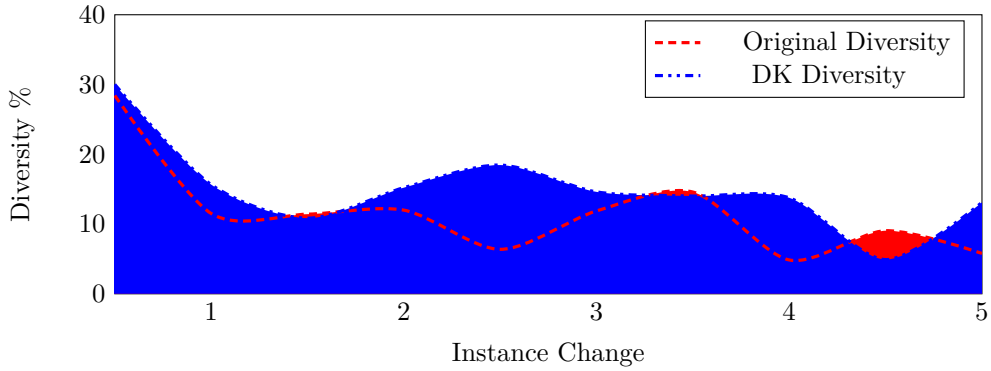


Figure 6.10: Average Level of Phenotypic Diversity Throughout Execution for a typical run of tartarus instance of size n=8, C=20 over 100 generations.

Figures 6.9 and 6.10 show the average level of phenotypic diversity for the time intervals C=15 and C=20 respectively. These results indicate that the impact of the reported fitness modification on the level of diversity present in the population is still present at larger time intervals.

Table 6.1 shows the average results from these experiments for both GP systems utilising DK fitness bias and canonical GP systems. The observed delta upon

instance change in the average population fitness, for one generation: +**1**, and five generations: +**5** after the change, is shown - illustrating the impact of the instance change on the average population fitness.

Table 6.1: Observed Average Population Fitness Delta for Canonical and DK Bias Systems.

		Canonical	95%CI ±	**DK Bias**	95%CI ±
C = 10	+1	0.9589	0.00145	**0.7644**	0.00134
	+5	0.7400	0.00145	**0.5722**	0.00134
C = 15	+1	**1.1322**	0.00106	1.1956	0.00116
	+5	0.9856	0.00106	**0.9500**	0.00116
C = 20	+1	1.2644	0.00170	**1.2578**	0.00154
	+5	1.0867	0.00170	**0.9440**	0.00154
C = 50	+1	1.3656	0.00074	**1.2722**	0.00091
	+5	1.4122	0.00074	**1.1733**	0.00091

The GP system utilising DK fitness bias is able to recover fitness performance faster than the canonical GP system, under change, with an average difference of 6.18% for +**1** and 12.36% for +**5**, across the 4 time intervals, C. Supporting the supposition that DK fitness bias increases the robustness of a GP system.

Additionally, Table 6.2 and Table 6.3 shows the delta in average population fitness for GP systems under instance change; utilising both DK fitness bias and fitness sharing, across 100 Tartarus instances of size $n = 8$, for distance k=1 and k=2 respectively. In fitness sharing, individuals close to each other [87] in the population share fitness scores. In the experiments, the fitness score of an individual was calculated both by averaging [78] and derating [88].

The k value is calculated using the ternary distance, as illustrated in Section 5.1, in order to find the surrounding individuals in the A_F – A_L – A_R allele space. Figure 6.11 illustrates the locations of neighbouring individuals at distance k=1, for use in fitness sharing.

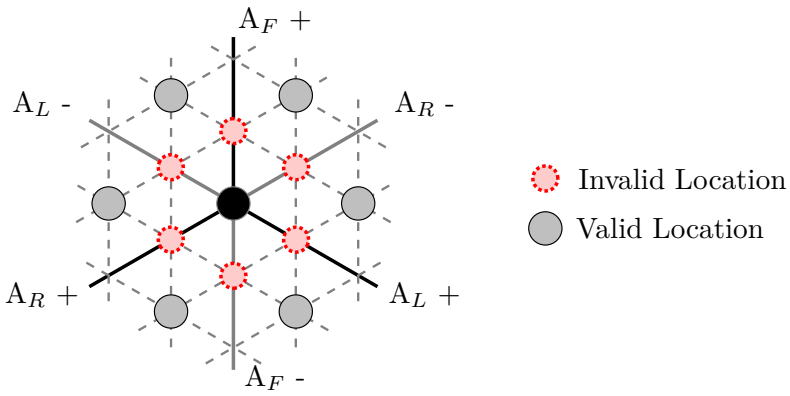


Figure 6.11: Valid and Invalid neighbourhood positions

The composition of alleles present in an individual are of a fixed length. Therefore, not all intersections, and their subsequent alleles, present in the space are valid.

Table 6.2: Observed Average Population Fitness Delta for DK Bias and Fitness Sharing (k=1) GP Systems.

		Averaged	Derated			
		Fitness Sharing k=1		95%CI ±	**DK Bias**	95%CI ±
C = 10	+1	0.8256	1.0613	0.00145	0.7644	0.00134
	+5	0.8747	1.1197	0.00145	0.5722	0.00134
C = 15	+1	0.9444	1.0063	0.00106	1.1956	0.00116
	+5	0.8390	0.9731	0.00106	0.9500	0.00116
C = 20	+1	0.9200	1.0032	0.00170	1.2578	0.00154
	+5	0.9976	1.0124	0.00170	0.9440	0.00154
C = 50	+1	1.0022	0.9752	0.00074	1.2722	0.00091
	+5	0.9901	0.9862	0.00074	1.1733	0.00091

Table 6.3: Observed Average Population Fitness Delta for DK Bias and Fitness Sharing (k=2) GP Systems.

		Averaged	Derated			
		Fitness Sharing k=2		95%CI ±	**DK Bias**	95%CI ±
C = 10	+1	0.4425	1.1181	0.00145	0.7644	0.00134
	+5	1.4240	0.9614	0.00145	0.5722	0.00134
C = 15	+1	1.0269	1.0069	0.00106	1.1956	0.00116
	+5	1.1637	1.0440	0.00106	0.9500	0.00116
C = 20	+1	0.9619	0.9950	0.00170	1.2578	0.00154
	+5	0.9470	0.9560	0.00170	0.9440	0.00154
C = 50	+1	1.0070	0.9839	0.00074	1.2722	0.00091
	+5	1.1046	0.9853	0.00074	1.1733	0.00091

From the results in Tables 6.2 and 6.3 it can be seen that the fitness sharing approach, both at a distance of k=1 and k=2, has broadly similar performance to the DK fitness bias. When the instance was updated at a shorter time interval, C=10 and C=15, the averaged fitness sharing approach appears to outperform the derated approach. However, when the frequency of change is reduced, C=50, with a longer time interval between updates, the derated approach returns stronger performance.

6.6 Conclusion

We proposed and implemented a novel fitness bias, inspired by the cognitive biases observed in the Dunning–Kruger effect. The use of the *reported* fitness values, as a result of utilising the DK fitness bias, lead to an approximate 10%

increase in the long-term level of population diversity.

Subsequently, when Tartarus instances were changed regularly at varying time intervals C, an increase in robustness was observed. The GP systems utilising the proposed DK fitness bias were able to recover performance faster and with a greater magnitude on average, compared to GP systems utilising canonical fitness evaluation.

In relation to **RQ3**, we can conclude that it is possible to influence the diversity and robustness of a population by utilising a simple fitness bias technique, inspired by the Dunning–Kruger effect.

Chapter 7

Conclusion

‘ ’

Chapter Contents

7.1 Conclusion

The research presented in this thesis has contributed to the literature and knowledge of genetic programming, by focusing on mechanisms by which to improve solution performance. Through the establishment of the Tartarus problem as a benchmark for use in GP (Ch. 4), the introduction of a self-adaptive crossover operator (Ch. 5) and a Dunning–Kruger inspired fitness bias (Ch. 6).

The results of the research demonstrate that the use of a self-adaptive crossover operator lead to an increase solution performance when utilised in a GP system, for the Tartarus problem and the Santa-Fe Trail problem. It was shown that the introduction of the Dunning-Kruger inspired fitness bias was able to effectively increase the level of long-term diversity present in a GP population - subsequently, leading to an increase in robustness for the Tartarus problem.

7.1.1 Thesis Contributions

The thesis has made the following contributions:

In relation to **RQ1**:

The Tartarus Problem is presented as a genetic programming benchmark, presenting a new improved instance evaluation mechanism. Guidance is provided for tuning the difficulty of generated Tartarus instances.

In genetic programming it is important that the structure of the reward mechanism matches the evolutionary processes. The rewarding of part-complete and

piecemeal solutions is key, so that better solutions can evolve gradually over time. In the canonical Tartarus problem, only the agents who completed the entire movement of a block from its starting position to the environment boundary are successful. This binary success or fail is considered to be too coarse to be practically useful, when the instance sizes are increased above the canonical instance of size n=6.

In this thesis we proposed an updated and improved Tartarus instance evaluation mechanism, rewarding states according to how close they are to a desired final state. This is done by measuring the aggregate distance between the blocks in the environment and the environment boundary. The improved evaluation mechanism is therefore able to reward agents who have made significant progress towards an optimal solution, but who would have scored 0 with the canonical Tartarus evaluation mechanism.

The increase in the granularity of the evaluation mechanism provided the ability for a greater degree of analysis to be carried out on Tartarus instances. This allowed for a set of baseline Tartarus values to be produced, culminating in a reference guide for creating Tartarus instances of sizes up to n=32. From this analysis it is possible to predict the difficulty of a given Tartarus instance, a range of parameter values and their associated difficulty level estimates are provided, allowing for the difficulty of a Tartarus instance to be tuned. The Tartarus problem has been established as a suitable benchmark problem by satisfying the characteristics outlined in Section 3.3.2.

In relation to **RQ2**:

A novel self-adaptive crossover operator is presented for use in a GP system, providing a continuous opportunity for parameter modifications to be made at runtime, and leading to an increase in solution performance.

Self-adaption aims at biasing the distribution of individuals in the population towards the areas of the search space which are more appropriate, producing effective solutions. This is achieved by means of setting and adjusting control parameters present in the GP system. A desired outcome of self-adaptive systems is to find ways in which to improve performance whilst maintaining the overall efficiency of the system.

We provided a novel self-adaptive crossover operator for use in GP, capable of modifying the control parameters at runtime, as-and-when they are deemed necessary. The proposed operator provides a continuous opportunity for modifications to be made at runtime, rather than sticking to a rigid, deterministic update schedule. This increase in flexibility allows for more effective modifications to be made. As the process of optimising a self-adaptive system is a dynamic problem in itself, the control parameters thought optimal at the start of, or during a run, may end up being unsuitable later on. Therefore, an operator which is able to offer a continuous opportunity to trigger modifications is preferable.

The introduction of the proposed self-adaptive crossover operator into the GP system, lead to an overall average increase in solution performance of approximately 15% for the Tartarus problem, and 10% for the Santa-Fe problem.

In relation to **RQ3**:

A novel fitness bias, inspired by the cognitive bias observed in the Dunning–Kruger effect is implemented in a GP system. As a result, an increase in population diversity, and subsequently solution robustness, was observed.

Robustness is referred to as a characteristic of a candidate solution whereby the performance is not diminished despite perturbations in the environmental parameters or constraints. A solution that is able to conserve utility under these changes is said to be robust. The diversity of a population has an impact on the robustness of the population: populations with a low level of diversity, homogenous populations, are less likely to be able to withstand large changes in the instance or environment being tested.

The Dunning–Kruger effect outlines cognitive biases observed in real-world psychological experiments, concluding that individuals with a low level of ability mistakenly over-estimate their performance whilst individuals with a high level of ability will over-estimate their performance.

In this thesis we proposed and implemented a novel fitness distribution bias, inspired by the cognitive biases observed in the Dunning–Kruger effect. The simulated DK bias modified the fitness scores of individuals based on their performance relative to the rest of the population, in a manner similar to the observed real-life DK bias. These updated fitness scores are then used for the selection of individuals for the next generation in the GP system. An approximate 10% increase in the level of long-term diversity was observed in the GP population.

When tartarus instances where changed regularly at varying time intervals, an increase in robustness was observed. The GP systems utilising the DK fitness bias were able to both recover performance faster and with a greater magnitude on average, compared to GP systems utilising canonical fitness evaluation.

7.1.2 Future Directions

The contributions presented in this thesis suggest potential avenues for future research to be conducted:

This thesis has provided understanding as to how the Tartarus problem can be considered a suitable benchmark problem for use in GP, further work should be concentrated on assembling a collective suite of benchmark problems. Analysis would be conducted, determining how performance is measured across multiple different benchmark problems, in order to assess multiple characteristics and facets of a candidate solution. It should be possible to attain a more detailed and accurate determination of the true performance of approaches using a suite of benchmark problems, opposed to a singular benchmark problem.

The Dunning–Kruger inspired fitness bias was shown to be successful at increasing the diversity present in a GP population, subsequently leading to an increase in robustness. Following this, a set of experiments comparing the effectiveness of other diversity management techniques and the proposed DK fitness bias should be carried out. This would allow for in-depth understanding of methods for improving the robustness of GP solutions.

Bibliography

[1] W. B. Langdon, *Genetic programming and data structures: genetic programming+ data structures= automatic programming!*, vol. 1. Springer Science & Business Media, 2012.

[2] D. R. Barstow, "Domain-specific automatic programming," *IEEE Transactions on Software Engineering*, no. 11, pp. 1321–1336, 1985.

[3] R. Balzer, "A 15 year perspective on automatic programming," *IEEE Transactions on Software Engineering*, no. 11, pp. 1257–1268, 1985.

[4] C. Darwin, *On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life.* London: John Murray, 1859.

[5] T. Griffiths and A. Ekárt, "Improving the tartarus problem as a benchmark in genetic programming.," in *Proceedings of the 20th European Conference on Genetic Programming -EuroGP'17* (J. e. a. McDermott, ed.), pp. 278–293, Springer, 2017.

[6] A. Teller, "The evolution of mental models," *Advances in Genetic Programming*, pp. 199–217, 1994.

[7] D. White, J. McDermott, M. Castelli, L. Manzoni, B. Goldman, G. Kronberger, W. Jáskowkski, U. O'Reilly, and S. Luke, "Better gp benchmarks: Community survey results and proposals.," *Genetic Programming and Evolvable Machines*, vol. 14, no. 1, pp. 3–29, 2013.

[8] T. Bäck., D. Fogel., and Z. Michalewicz, *Handbook of Evolutionary Computation*, pp. C7.1:1–C7.1:15. Oxford University Press, 1997.

[9] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic programming IV: Routine human-competitive machine intelligence*, vol. 5. Springer Science & Business Media, 2006.

[10] J. Liu and K. C. Tsui, "Toward nature-inspired computing," *Communications of the ACM*, vol. 49, no. 10, pp. 59–64, 2006.

[11] J. R. Koza, D. Andre, M. A. Keane, and F. H. Bennett III, *Genetic programming III: Darwinian invention and problem solving*, vol. 3. Morgan Kaufmann, 1999.

[12] J. Koza, *Genetic programming: On the programming of computers by means of natural selection.* 1992.

[13] J. R. Koza *et al.*, *Genetic programming II*, vol. 17. MIT press Cambridge, MA, 1994.

[14] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, 2004.

[15] T. Griffiths and A. Ekárt, "Improving the effectiveness of genetic programming using self-adaptation.," in *Proceedings of The 2nd International Symposium on Artificial Life and Intelligent Agents - ALIA'16*, vol. 732, pp. 97–102, CCIS, Springer, 2016.

[16] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the first international conference on genetic algorithms*, pp. 183–187, 1985.

[17] A. Ekárt and S. Z. Németh, "A metric for genetic programs and fitness sharing," in *European Conference on Genetic Programming*, pp. 259–270, Springer, 2000.

[18] O. Krauss, H. Mössenböck, and M. Affenzeller, "Dynamic fitness function for genetic improvement in compilers and interpreters," 2018.

[19] T. Soule, J. A. Foster, and J. Dickinson, "Code growth in genetic programming," in *Proceedings of the 1st annual conference on genetic programming*, pp. 215–223, MIT Press, 1996.

[20] S. Luke, "Modification point depth and genome growth in genetic programming," *Evolutionary Computation*, vol. 11, no. 1, pp. 67–106, 2003.

[21] M. Brameier and W. Banzhaf, *Linear Genetic Programming*. Springer, 2007.

[22] J. Miller and P. Thomson, "Cartesian genetic programming.," in *Proceedings Genetic Programming: European Conference - EuroGP 2000*, pp. ?–?, LNCS, Springer, 2000.

[23] R. Mckay, N. Hoai, P. Whigham, Y. Shan, and M. O'neill, "Grammar-based genetic programming: a survey.," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 365–396, 2010.

[24] W. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*. San Francisco: Morgan Kaufmann, 1998.

[25] W. Langdon, R. Poli, N. McPhee, and J. Koza, "Genetic programming: An introduction and tutorial, with a survey of techniques and applications," *Studies in Computational Intelligence*, vol. 115, pp. 927–1028, 2008.

[26] M. F. Brameier and W. Banzhaf, "A comparison with tree-based genetic programming," *Linear Genetic Programming*, pp. 173–192, 2007.

[27] W. Banzhaf, P. Nordin, R. Keller, and F. Francome, *Genetic programming - An introduction. On the automatic evolution of computer programs and it application*. San Francisco: Morgan Kaufmann, 1998.

[28] S. L. Harding, J. F. Miller, and W. Banzhaf, "Self-modifying cartesian genetic programming," in *Cartesian Genetic Programming*, pp. 101–124, Springer, 2011.

[29] J. Husa and R. Kalkreuth, "A comparative study on crossover in cartesian genetic programming," 2018.

[30] W. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, vol. 4 of *Genetic Programming*. Springer, 2003.

[31] T. Back, "Selective pressure in evolutionary algorithms: a characterization of selection mechanisms," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 57–62, 1994.

[32] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[33] B. L. Miller, D. E. Goldberg, *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[34] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[35] H. MÃijhlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm i. continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.

[36] J. Page, R. Poli, and W. B. Langdon, "Mutation in genetic programming: A preliminary study," in *European Conference on Genetic Programming*, pp. 39–48, Springer, 1999.

[37] B. Wie and D. S. Bernstein, "Benchmark problems for robust control design," *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 5, pp. 1057–1059, 1992.

[38] J. McDermott, D. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jáskowski, K. Kraweic, R. Harper, K. DeJong, and U. O'Reilly, "Genetic programming needs better benchmarks.," in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation - GECCO '12*. (T. e. a. Soule, ed.), pp. 791–798, Springer, 2012.

[39] O. Mersmann, M. Preuss, M. Trautmann, H. Bischl, and C. Weihs, "Analyzing the bbob results by means of benchmarking concepts.," *Evolutionary Computation*, vol. 23, no. 1, pp. 161–185, 2015.

[40] D. Whitley, S. Rana, J. Dzubera, and E. Mathias, "Evaluating evolutionary algorithms.," *Artificial Intelligence*, vol. 85, pp. 245–276, 1996.

[41] G. C. Cawley and N. L. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 2079–2107, 2010.

[42] S. Thrun and L. Pratt, *Learning to Learn*. Springer, 1998.

[43] H. Iba and H. de Garis, "Extending genetic programming with recombinative guidance," *Advances in genetic programming*, vol. 2, pp. 69–88, 1996.

[44] A. Eiben., Z. Michalewicz., M. Schoenauer., and J. Smith., "Parameter control in evolutionary algorithms.," in *Parameter Setting in Evolutionary Algorithms - SCI* (C. L. F.G. Lobo. and Z. Michalewicz., eds.), vol. 54, pp. 19–46, 2007.

[45] S. Meyer-Nieberg and H.-G. Beyer, *Self-Adaptation in Evolutionary Algorithms*, vol. 54, pp. 47–75. 04 2007.

[46] R. Rosenberg., *Simulation of genetic population with biochemical properties.* PhD thesis, Univ. Michigan, 1967.

[47] J. Bagley., *The Behaviour of Adaptive Systems which employ Genetic and Correlation Algorithms.* PhD thesis, Univ. Michigan, 1967.

[48] T. Back, "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm," in *Proc. 2nd Conference of Parallel Problem Solving from Nature, 1992*, Elsevier Science Publishers, 1992.

[49] J. Schaffer. and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp. 36–40, 1987.

[50] T. Griffiths. and A. Ekárt, "Self-adaptive crossover in genetic programming: The case of the tartarus problem.," in *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature - PPSN XV*, pp. 236–246, Springer, 2018.

[51] S. Kirkpatrick., C. Gelatt., and M. Vecchi, "Optimisation by simulated annealing.," *Science.*, vol. 220, pp. 671–680, 1983.

[52] A. Qin. and P. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization.," in *Proceedings of the 2005 Congress on Evolutionary Computation*, vol. 2, pp. 23–32, IEEE, 2005.

[53] J. Hesser. and R. Manner., "Towards an optimal mutation probablity in genetic algorithms.," in *Proceedings of the 1st Conference on Parallel Problem Solving from Nature - PPSN I* (H. Schwefel and R. Männer., eds.), pp. 23–32, Springer, 1991.

[54] N. Hansen., A. Ostermeier., and A. Gawelczyk., "On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation.," in *Proceedings of the 6th International Conference on Genetic Algorithms - ICGA'95* (L. Eshelman., ed.), pp. 57–64, Morgan Kaufmann, 1995.

[55] R. Hinterding., Z. Michalewicz., and T. Peachey, "Self-adaptive genetic algorithm for numeric functions," pp. 420–429, 1996.

[56] T. Bäck, "The interaction of mutation rate, selection and self-adaptation within a genetic algorithm," in *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature - PPSN II*, pp. 85–94, 1992.

[57] D. Dang. and P. Lehre, "Self-adaptation of mutation rates in non-elitist populations," 2016.

[58] P. Angeline., "Adaptive and self-adaptive computations," in *Computational Intelligence: A Dynamic Systems Perspective*, pp. 152–163, IEEE Press, 1995.

[59] P. Taylor and R. Lewontin, "The genotype/phenotype distinction," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, summer 2017 ed., 2017.

[60] Y. Jin and B. Sendhoff, "Trade-off between performance and robustness: an evolutionary multiobjective approach," in *international conference on Evolutionary Multi-Criterion Optimization*, pp. 237–251, Springer, 2003.

[61] C. Barrico and C. H. Antunes, "Robustness analysis in multi-objective optimization using a degree of robustness concept," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 1887–1892, IEEE, 2006.

[62] Y. Jin, J. Branke, *et al.*, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on evolutionary computation*, vol. 9, no. 3, pp. 303–317, 2005.

[63] D. Dunning, "The dunning–kruger effect: On being ignorant of one's own ignorance," in *Advances in experimental social psychology*, vol. 44, pp. 247–296, Elsevier, 2011.

[64] J. Kruger and D. Dunning, "Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments.," *Journal of personality and social psychology*, vol. 77, no. 6, p. 1121, 1999.

[65] D. Dunning, K. Johnson, J. Ehrlinger, and J. Kruger, "Why people fail to recognize their own incompetence," *Current directions in psychological science*, vol. 12, no. 3, pp. 83–87, 2003.

[66] J. Huxley, *Evolution, the Modern Synthesis*. G. Allen & Unwin Limited, 1942.

[67] P. Alberch, "From genes to phenotype: Dynamical systems and evolvability.," *Genetica*, vol. 84, pp. 5–11, 1991.

[68] M. Pigliucci, "Genotype-phenotype mapping and the end of the 'genes as a blueprint' metaphor.," *Philosophical Transactions of the Royal Society B*, vol. 365, pp. 557–566, 2010.

[69] M. Pigliucci, "Genotype–phenotype mapping and the end of the âĂŸgenes as blueprint-âĂŹmetaphor," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 365, no. 1540, pp. 557–566, 2010.

[70] "Increasing genetic programming robustness using simulated dunning-kruger effect,"

[71] D. Dunham, "Robustness of genetic algorithm solutions in resource levelling," in *IEEE Systems and Information Engineering Design Symposium*, 2015.

[72] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments–a survey.," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[73] S. Yang, Y.-S. Ong, and Y. Jin, *Evolutionary computation in dynamic and uncertain environments*, vol. 51. Springer Science & Business Media, 2007.

[74] G. Oster and P. Alberch, "Evolution and bifurcation of developmental programs," *Evolution*, pp. 444–459, 1982.

[75] T. Hu and W. Banzhaf, "Neutrality, robustness, and evolvability in genetic programming," in *Genetic Programming Theory and Practice XIV*, pp. 101–117, Springer, 2018.

[76] A. Wagner, "Robustness and evolvability: a paradox resolved," *Proceedings of the Royal Society B: Biological Sciences*, vol. 275, no. 1630, pp. 91–100, 2007.

[77] N. F. McPhee and N. J. Hopper, "Analysis of genetic diversity through population history," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pp. 1112–1120, Morgan Kaufmann Publishers Inc., 1999.

[78] A. Ekárt and S. Z. Németh, "Maintaining the diversity of genetic programs," in *European Conference on Genetic Programming*, pp. 162–171, Springer, 2002.

[79] P. D'haeseleer, "Context preserving crossover in genetic programming," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 256–261, IEEE, 1994.

[80] J. P. Rosca, "Genetic programming exploratory power and the discovery of functions.," in *Evolutionary Programming*, pp. 719–736, Citeseer, 1995.

[81] D. Jackson, "Phenotypic diversity in initial genetic programming populations," in *European Conference on Genetic Programming*, pp. 98–109, Springer, 2010.

[82] S.-Y. Lu, "A tree-to-tree distance and its application to cluster analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 219–224, 1979.

[83] S. M. Selkow, "The tree-to-tree editing problem," *Information processing letters*, vol. 6, no. 6, pp. 184–186, 1977.

[84] K.-C. Tai, "The tree-to-tree correction problem," *Computer Algorithms: String Pattern Matching Strategies*, vol. 55, p. 208, 1994.

[85] O. J. Mengshoel and D. E. Goldberg, "The crowding approach to niching in genetic algorithms," *Evolutionary computation*, vol. 16, no. 3, pp. 315–354, 2008.

[86] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[87] H. Berg, "Fitness sharing based on angular distances," in *2009 Fifth International Conference on Natural Computation*, vol. 4, pp. 237–243, IEEE, 2009.

[88] P. S. Oliveto, D. Sudholt, and C. Zarges, "On the benefits and risks of using fitness sharing for multimodal optimisation," *Theoretical Computer Science*, vol. 773, pp. 53–70, 2019.

[89] K. Balakrishnan and V. Honavar, "On sensor evolution in robotics," in *Proceedings of the First International Conference on Genetic Programming*, pp. 455–460, Citeseer, 1996.

Appendix

Experimental Parameterisation

Outlined below is a summary of the parameter setups utilised in the thesis, with symbolic and numeric [44] parameters provided.

In Table 7.1 the parameter setups utilised in the *Tartarus Baseline* and *Improved Tartarus State Evaluation* experimentation from Chapter 4 is represented by T_{ps}, the setup for the *Tartarus Case Study* from Chapter 5 is represented by TC_{ps} and the setup for the *Simulated Dunning-Kruger* experimentation in Chapter 6 is represented by DK_{ps}.

Table 7.1: Experimental Parameter Setup

		T_{ps}	TC_{ps}	DK_{ps}
Symbolic	Recombination	1-point	1-point	sDK
	Parent Selection	Truncation	Truncation	Truncation
	Mutation	Single Point	Single Point	Single Point
	Survivor Selection	(μ, λ)	(μ, λ)	(μ, λ)
Numeric	Mutation Rate	0.05	0.05	0.05
	Mutation Step Size	0.02	0.02	0.02
	Crossover Rate	0.5	0.5	0.5
	Population Size p_x	50	50	100
	Offspring Size	p_x	p_x	$2p_x$
	Proportion Point *pp*	0.4	0.4	0.4

For *Parent Selection* a method of truncation is used - whereby all individuals are ordered according to their fitness scores and a proportion of the fittest individuals are selected. The proportion of individuals selected from the population is determined by the *Proportion Point*, *pp* value in Table 7.1. The selection pressure can be altered by selecting a different proportion point.

(in the range $0 < pp \leq 1$, where *pp* selects at least 2 individuals.)

Throughout each generation the total population size is maintained, determined by *Population Size*, p_x in Table 7.1.

Therefore, calculating the (μ, λ) values, using T_{ps} as an example:

$\mu = p_x \times pp = 20$ and,

$\lambda = p_x = 50$,

and for the Dunning-Kruger setting:

$\mu = p_x \times pp = 40$ and,

$\lambda = p_x = 100$.

Example GP Representation

In previous work on the Tartarus problem it was noted that allowing the agent to alter the placement and number of sensors, could lead to changes in overall performance [89].

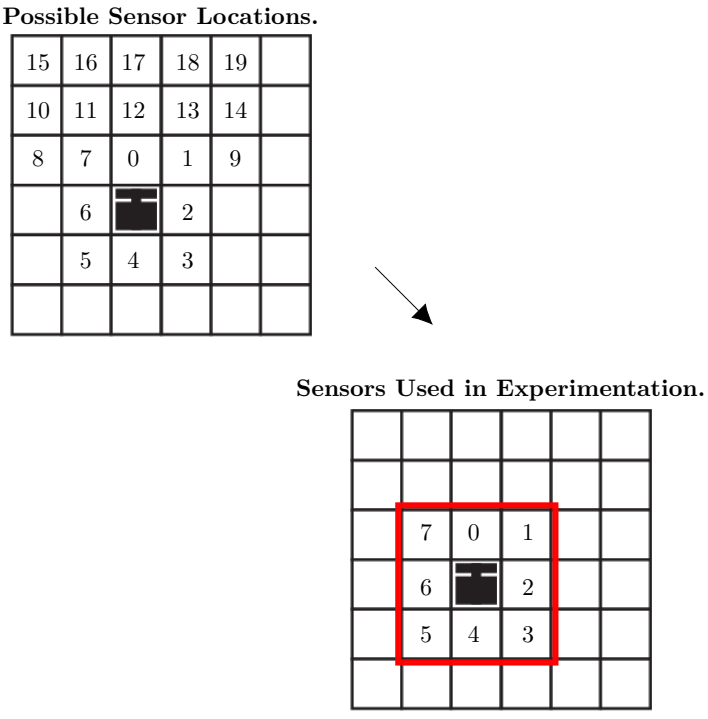


Figure 7.1: Dozer Agent Sensor Locations

However, for the experiments outlined throughout the thesis, the environmental representations (i.e. sensor placement and number) were fixed. Figure 7.1 shows the difference between the largest possible environmental representation and the canonical environmental representation utilised in the thesis experimentation.

The smaller environmental representation was used in this thesis as it allowed for effective comparisons to be made with previous and canonical work on the Tartarus problem.

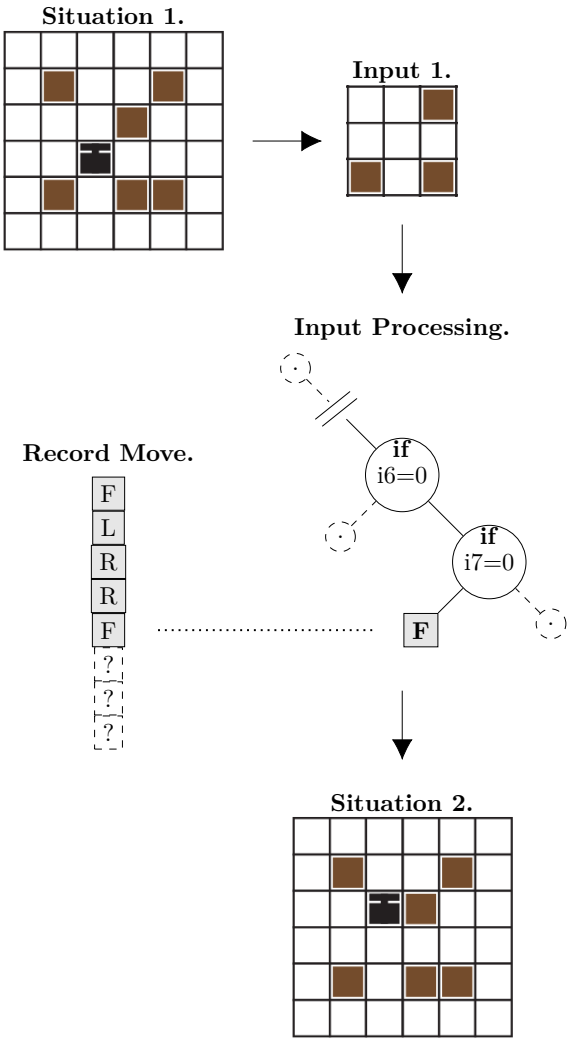


Figure 7.2: Example GP Representation

Figure 7.2 illustrates excerpts from a canonical GP instance – illustrating how the structural characteristics of the representation is reflected in the agent interaction and candidate solution. A representative GP agent-environment interaction is shown with the agent receiving input from the environment surrounding it (using the environmental representation outlined in Figure 7.1). The sensor inputs are boolean values, with **0** used for an empty gridsquare and **1** for an occupied gridsquare - with the edges of the environment treated as occupied gridsquares.

This input is processed and the decision tree corresponding to the agent is traversed. The decision trees are comprised of internal nodes which can contain a mixture if-else boolean operators relating to the status of the sensors, and leaf nodes corresponding to the terminal set of moves, chosen from the following actions: **1) Turn left**, **2) Turn right**, **3) Move forwards one square** this move is then executed by the agent and recorded.

As shown in Figure 7.2 the agent executed *move forwards one square* - f, as an outcome of traversing the decision tree - with the decisions based on whether or not there is a block present in input 6 and 7 shown.

This process of using the input sensor values in the current situation, executing the decision tree to find the corresponding action, recording the action and executing the action is repeated until the maximum number of allowed moves m is reached. The run is then completed and the instance score is calculated and the corresponding resultant fitness score is assigned to the agent. Through the recom-

bination process the fitter individuals, and their corresponding decision trees and variable values, will likely survive leading to an increase in overall performance.

The aforementioned summary of the parameter setups utilised in this thesis, outlined the process by which the agent-environment interaction occurs and the inputs are processed to select the next movement operation at runtime.