Improving performance of genetic algorithms by using novel fitness functions

Improving Performance of Genetic Algorithms by Using Novel Fitness Functions

by

Jason Cooper

A Doctoral Thesis

Submitted in partial fulfilment of

the requirements for the award of

Doctor of Philosophy

of

Loughborough University

January 20, 2006

Abstract

This thesis introduces Intelligent Fitness Functions and Partial Fitness Functions both of which can improve the performance of a genetic algorithm which is limited to a fixed run time.

An Intelligent Fitness Function is defined as a fitness function with a memory. The memory is used to store information about individuals so that duplicate individuals do not need to have their fitness tested. Different types of memory (long and short term) and different storage strategies (fitness based, time base and frequency based) have been tested. The results show that an intelligent fitness function, with a time based long term memory improves the efficiency of a genetic algorithm the most.

A Partial Fitness Function is defined as a fitness function that only partially tests the fitness of an individual at each generation. Thus only promising individuals get fully tested. Using a partial fitness function gives the genetic algorithm more evolutionary steps in the same length of time as a genetic algorithm using a normal fitness function. The results show that a genetic algorithm using a partial fitness function can achieve higher fitness levels than a genetic algorithm using a normal fitness function.

Finally a genetic algorithm designed to solve a substitution cipher is compared to one equipped with an intelligent fitness function and another equipped with a partial fitness function. The genetic algorithm with the intelligent fitness function and the genetic algorithm with the partial fitness function both show a significant improvement over the genetic algorithm with a conventional fitness function.

Keywords : Genetic Algorithms, Fitness Functions, Efficiency, Intelligent Fitness Functions, Partial Fitness Functions

Acknowledgements

I would like to thank the following people for helping me create this thesis and for all the help and support they have provided.

First and foremost I have to thank Dr. Chris Hinde for the excellent supervision of this work and the many conversations unrelated to this work, but of great enjoyment nonetheless.

I would also like to thank my wife, Erica Cooper, and my newborn son for helping me to keep working at my thesis when I didn't always feel like it.

I must thank my parents who gave me the support when I needed it and without whom I would not even exist.

Thanks must go to Professor Paul Chung, who did an excellent job as my Director of Research and made me think two steps ahead. Thanks also to Dr. Roger Stone, while not my supervisor he was always there to give advice and support.

Thanks to Mick O'Doherty and the rest of the crew at the Maidenhead branch of Nortel Networks who helped finance my first two years of research.

Thanks to Dr Mark Withall and the rest of the research students in the Computer

Science department, without whom many an interesting problem would have passed me by.

Thanks to everyone who has proof read my thesis, without whom many a spelling mistake would still be left in.

Finally a special thanks to Mr Mark Brill for providing an endless supply of DVDs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Survey

1.1 Introduction

This Thesis introduces two methods designed to help improve the efficiency and performance of GAs. These methods are then investigated to examine the effect that they have on GAs.

1.1.1 Aims and Objectives

The aim of this Thesis is to introduce new methods to improve the performance of GAs.

The following are a list of objectives for this Thesis.

- Assess the effect population sizes have on the performance and efficiency of GAs running within a fixed length of time with a slow fitness function.

- Introduce new methods for improving the performance and efficiency of GAs with a slow fitness function.
- Investigate the effects that the introduced methods have on GAs running within a fixed length of time.
- Investigate the improvement provided by the methods to a GA with a slow fitness function.

1.1.2 Content Summary

The content of this Thesis is as follows

Chapter 1 - An introduction to this Thesis, evolution and GAs.

Chapter 2 - A short description of the work that posed the question addressed by this Thesis. The work consists of the evaluation of the effects different configurations of packet options have on the time it takes a packet to travel across the Internet.

Chapter 3 - Investigates the effect that population size and the level of elitism has on the efficiency and performance of GAs. Section 3.3 describes the test problems used. Section 3.5.1 examines the effects that population sizes have on the performance of a GA, while section 3.5.4 examines the effects that population sizes have on the efficiency of a GA. Finally section 3.5.5 examines the effects that the level of elitism has on a GA.

Chapter 4 - Introduces Intelligent Fitness Functions and investigates the effects they have on a GA's efficiency. GAs equipped with a selection of intelligent fitness functions, each with a different configuration, are compared to each other and a GA equipped with a standard fitness function.

The performance of a GA equipped with an intelligent fitness function is shown to be better than that of a GA equipped with a standard fitness function.

Chapter 5 - Introduces Partial Fitness Functions and investigates the effects they have on a GA's efficiency. GAs equipped with differently configured partial fitness functions are compared to each other as well as a GA equipped with a standard fitness function.

The levels of fitness achieved by a GA equipped with a partial fitness function are shown to be better than the fitness achieved by a GA equipped with a standard fitness function.

Chapter 6 - Documents the results of using a GA with a standard fitness function, a GA with an intelligent fitness function and a GA with a Partial Fitness Function, to break a substitution cipher.

Both the GA equipped with an intelligent fitness function and the GA equipped with a partial fitness function perform better than the GA with a standard fitness function.

Chapter 7 - A discussion of the work contained in this Thesis and conclusions drawn from it.

1.2 Evolution

Evolution is the process that allows species to improve their ability to survive in their environment. Through evolution species can adapt to changes in their environment. As species evolve they can become more complex and significantly different from the original species, these are then classed as a new species. Evolution works by incorporating changes, usually small, that are advantageous to the species and removing attributes that are not. This process works because individuals with the advantageous attributes survive longer in their environment to reproduce, whereas the individuals without the attributes, or with worse attributes, die earlier or are unable to attract mates [28].

An extreme example of this is a bacteria that has infected someone making them ill. If the person takes drug A, most of the bacteria that are susceptible to drug A will die. Therefore, mainly bacteria immune to drug A will be left to reproduce and therefore most of the offspring will be immune. The person then becomes ill again when the bacteria has reproduced enough and so takes drug A, with little or no effect. The person then takes drug B, killing most of the bacteria susceptible to drug B leaving mainly those immune to both drug A and drug B. These then reproduce to give offspring that are immune to both drug A and drug B. And so on.

Evolution can be split into two processes, natural selection and inheritance. Natural selection can be viewed as a high level process as it deals with the interaction between a species and its environment. Whereas, inheritance can be viewed as a low-level process as it works on a genetic level of each individual. The following sections describe in more detail these two processes and how they help to make evolution work.

1.2.1 Natural Selection

The process of natural selection deals with the selection of which individuals from a population reproduce based on their fitness in their current environment. This process works because if an individual is not very fit it will die early or not have the ability to attract a mate. This is known as survival of the fittest.

There is a process known as unnatural selection that has been around for centuries. The major difference between natural and unnatural selection is that in unnatural selection the parents are chosen by people rather than nature. Unnatural selection is mainly used to exaggerate attributes in a species, for example, breeding racehorses, improving the quality of wool from sheep or breeding dogs for different purposes. Both natural selection and unnatural selection work for the same reasons. They both rely upon the fact that offspring tend to inherit attributes from their parents.

The concept of natural selection is commonly attributed to Charles Darwin [28], but it was also discovered independently at the same time by Alfred Russell Wallace [21]. Both had similar experiences that inspired their theories. Darwin on his voyage on the HMS Beagle, while Wallace an expedition to the Amazon. Both are believed to have been inspired by an essay written by Thomas Malthus [95]. In his work, Malthus argues that a population will always outgrow the available resources of an area, and hence competition between those occupying the same area will control the population. In essence, Malthus is saying that those in the population that are better able to acquire, or control, the available resources will prosper, whereas those who are less able will die early through lack of those resources. This in an early example of what would later become known as survival of the fittest. Prior to Darwin and Wallace, some people believed in the concept of evolution, however they had no process for it to work.

1.2.2 Inheritance and Genetics

Every cell in every living thing contains a blueprint showing how to build the organism. The blueprint is encoded in DNA. The blueprint in each cell is called the Genome. The genome consists of chromosomes and the chromosomes are made up of genes. Each gene defines an attribute of the organism. However, the attributes that are defined are dependent on the species the DNA is in. When a child organism is produced it will contain a combination of the genes of its parents.

Inheritance is the process that allows parents to pass on their attributes to their offspring. This is achieved because the offspring have their parents' genes. In asexual reproduction the genes of the offspring are only taken from one parent and so tend to be identical to their parents, although mutations do occur. In sexual reproduction the genes of the offspring are a combination of the genes of the parents and so the offspring inherit some attributes from each parent.

In humans each gene has one value from each parent. If both genes have the same value then that value is used, if they are different then one of the genes will be dominant over the other and the dominant value used [30]. Having two values for each gene is known as diploid [1]. The presence of a recessive gene in a diploid format enables a species to adapt to changes in the environment more quickly.

Richard Dawkins [30, 31, 32, 33] has written many books on this area. The groundwork for the area of genetics was laid by Gregor Mendel [98] who was a monk in the mid-19th century. However, his work was not widely known until the beginning of the 20th century, it consisted of experiments where he bred pea plants and documented the

[1] Having a single value for each gene is known as haploid while have three values for each gene is known as triploid.

results.

1.3 Genetic Algorithms

GAs have mainly been attributed to John Henry Holland but, as noted by David Fogel[40], other researchers were using methods very similar to GAs before Holland. Fraser [43, 44, 45, 46, 47], Bremermann et al. [11, 12, 13, 14, 15, 16, 17, 18] and Reed et al. [125] were all working on problem solving methods very similar to Holland's GAs. In fact the use of evolution to solve problems had been suggested in the 1940s by Alan Turing[144].

1.3.1 Genetic Algorithm Terminology

The following are terms that will be used in the rest of this Thesis when referring to GAs.

Genome - an encoded set of parameters

Individual - a member of the population consisting of a genome and if known the fitness level of the genome.

Genetic Operator - a function that takes a number of Genomes and produces a single Genome

Fitness Function - a function that takes a Genome and produces a fitness level that represents how well the parameters solve the problem.

1.3.2 Encoding the Solution

For a GA to tackle a problem the parameters for the algorithm need to be encoded in a form that the GA can manipulate (the genome) to produce new solutions that inherit traits from the parents used. While there are many methods for encoding the parameters in a GA there are two methods commonly used.

- Binary Encoding [67]
- Real Encoding [148]

Binary Encoding

The parameters for the solution are broken down into a binary string. When the solution is to be fitness tested it has to be decoded from the binary string back into the parameters for the solution. This has the advantages that all the genetic operators are easy to code and understand. It also means that core of the genetic algorithm can be used again and again to tackle different problems with only a need to modify the decoder and the fitness function. Holland [67] suggests that binary encoding should always be used, though it has been criticised by some [145, 3, 103].

Real Encoding

The GA operates directly on the parameters for the solution algorithm. No decoding is necessary for the fitness test, it just takes the parameters directly from the genome. This method has the advantage that no decoder is necessary but each GA has to be custom built for the problem it is tackling. Many researchers have used Real Encoding instead of the Binary Encoding suggested by Holland[67] and achieved good results to problems[103, 105, 140, 148].

Other Encoding Methods

There are many different methods that have been used for storing the parameters for a GA. These include the following

Trees - More commonly used in genetic programming [80].

Linear Program Encoding - An encoding method used to enable a standard GA to evolve programs [147].

1.3.3 Fitness Testing

The fitness test in a GA is the part which decides how well an individual solves a problem. If the genomes of the individuals are encoded using a binary encoding then the fitness function will have to decode them to obtain the parameters to be used.

1.3.4 Reproduction

After each generation has been fitness tested the GA has to produce a new generation. This is achieved through the use of genetic operators. The genetic operators used are the following.

- Selection

- Crossover
- Mutation

Selection Operator

This operator is used to select the parent individuals to be used to produce a new individual. To be effective it has to be biased towards selecting individuals with a high fitness, while still selecting a diverse collection of parents. There are two common methods used for selection strategies for the selection operator.

Roulette Wheel When the selection operator uses a roulette wheel selection method then the chance of an individual being chosen is directly proportional to the percentage of its fitness compared to the total fitness achieved by the generation. E.g. if an individual scored a fitness of 20 and the generation total was a fitness level of 100 then the chance of selecting the individual would be $\frac{1}{5}$.

Tournament When the selection operator uses a tournament selection method a number of individuals are selected from the population randomly and the one with the best fitness is then used as the selected individual.

Crossover Operator

Termed the distinguishing feature of GAs[29] this genetic operator takes two genomes and combines them to produce a new genome. Holland has stated that crossover is the main operator of GAs[67]. There are three commonly used methods.

- Single Point
- Multi Point
- Uniform

Single Point Single point crossover picks a break point along the genome and takes the genes from the first genome up to the break point and then uses the genes from the second genome to finish off the new genome. e.g.

```
| = Break point

Parent 1 : 0000000|000
Parent 2 : 1111111|111
-----------------------
Child    : 0000000|111
```

Multi Point Similar to the single point crossover but more than one break point is used. e.g.

```
| = Break point

Parent 1 : 00000|00|00
Parent 2 : 11111|11|11
-----------------------
Child    : 00000|11|00
```

Uniform Each gene to be used in the new genome is picked randomly from the relevant genes from the parents. e.g.

```
Parent 1 : 0000000000
Parent 2 : 1111111111
----------------------
Child    : 0110011101
```

Mutation Operator

The last operator used by GAs is the mutation operator. This operator takes a genome and then mutates random parts of it. So for a binary string each bit would have a small chance of being inverted. The chance of a bit being inverted is called the mutation rate and is usually set to $\frac{1}{n}$ where n is the length of the genome in its encoded form [51].

An example of mutation is as follows

```
Parent   : 0000000000
----------------------
Child    : 0100000001
```

Combination of the Operators

Goldberg [51] states that individually the operators are of no use. It is not until they are combined that any real benefit is encountered. Simply selecting the better individuals for the next generation would lead to a population consisting of the best individual from the first population. Crossover and mutation on their own do not improve the individuals, just changes them. But by combining the operators the GA is then capable of improving on each generation.[52]

Initial Populations

The initial population of a GA can be initialised by different methods.

- Random
- Seeded
- Mixed

Random The initial population is generated using a pseudo random number generator. This method's performance has been shown to be affected by the pseudo random number generator used by the GA [22, 101, 102], but it does keep the initial population mostly unbiased.

Seeded The initial population is created from a collection of known solutions to the problem [93]. This method gives the GA a population of good solutions to start with. However this can lead to a initial population that is biased in many ways, which can make it harder for the GA to find the better solutions.

Mixed The initial population is created from some seed individuals and the rest of the population is randomly generated. This has been shown to hinder the GA's abilities [114]. Though no explanation for this was provided it is probably due to the randomly generated individuals being vastly inferior to the seeded individuals.

Elitism

This is a process that enables a GA to carry a selection of a generation forward to the next generation without any changes. This is used to keep the best individuals encountered in the population. The degree of elitism dictates how many individuals are carried forward to the next generation and how many are replaced with new individuals. The higher the degree the more individuals carried forward and the fewer new individuals introduced. The individuals selected to be carried forward are always the ones with the highest fitness level.

Diploid

The majority of research into GAs has been aimed towards the haploid representation shown earlier in this chapter. Research into using diploid representation for GA, both with and without dominance, has shown that diploid shows an improvement over the haploid GAs when the environment being evolved in is unstable. Haploid GAs evolving in a stable environment have been shown to, at worst, perform to the same level as diploid GA [152].

Approximation

The computational effort required to test the fitness of a solution can be large. Grefenstette et al.[56] has shown for some of these problems better results can be obtained by the GA by using an approximation to a fitness test rather than a full evaluation. The advantages of using an approximation is that the time of each fitness test drops and hence more fitness tests can be performed in the same length of time. The disadvantages of using an approximation though is that accuracy is sacrificed for speed. Approximations have been used with GAs to tackle engineering problems like aircraft design[122] and predicting the feedback of polynomial LFSRs[63]. Jin et al. have produced two surveys on the area of approximations in GAs[71, 72].

There are two basic approaches to approximation in GAs.

- Functional Approximation
- Problem Approximation

Functional Approximation involves constructing an alternate and explicit expression for the fitness function[71, 72]. Using this expression instead of the full fitness function will decrease the time needed by the GA to assess each individuals fitness.

Problem Approximation replaces the problem being tackled with one that is approximately the same but easier for the GA to solve[71, 72]. As the new problem is easier for the GA to solve it can produce a good solution in fewer generations. This will decrease the length of time needed by the GA even if the new problem's fitness function takes the same length of time as the old problem's fitness function.

Approximation in GAs has mostly been applied to the following cases

Where the fitness function would be complex and time consuming. An example of complex fitness functions commonly approximated is in the area of structural design [57, 89, 79, 60, 108, 134, 113, 70]. When dealing with structures that must have a specific aerodynamic property computational fluid dynamics simulations would have to be carried out. These simulations can take over ten hours on a high-performance computer.

Where there is no explicit model to calculate the fitness. Evolving the design of art or the composition of music has a fitness function that depends on the user. Usually the system interacts with the user to get their opinion [142]. An approximation of the users opinions have been used to help [7, 73].

Where the environment is noisy. Without the use of approximation there are two common methods used to tackle the problem of noisy fitness functions. The first is to take multiple samples of the fitness and take the average as the actual fitness [37]. The alternate method used is to average the individual being tested with that of the individual located near to it in the search space [9].

However, with the use of a statistical model to estimate the fitness of neighbouring individuals, the computational cost can be reduced [133, 10].

Where the fitness function is multi-modal. An approximation can sometimes be created that smooths out the local minima and still has the global optimum in the same location [90].

There are two main concerns about using approximations in GAs. The first is that the GA should converge at the optimum or a near optimum of the conventional fitness function. The second concern is that the computational cost should be reduced as much as possible.

A GA using an approximate model can use one of three methods of evolution control. Evolution control is where some of the individuals are evaluated using the original fitness function rather than the approximation. This helps the GA to avoid false minima (where the approximation model predicts an optimum solution that does not exist in the original fitness function) [69]. The three methods of evolution control are the following.

No Evolution Control The approximate model is high-fidelity and as such does not need any evolution control [7, 124, 73].

Fixed Evolution Control There are two approaches to evolution control, individual based [57, 20] and generation based [120, 121].

With individual based evolution control some of the individuals in each generation are tested with the original fitness function. The individuals to be tested with the original fitness function can either be chosen using a random strategy or best strategy [69]. The best strategy re-evaluates the individual, with the best fitness from the approximation, using the original fitness function [57]. The random strategy selects the individuals to be re-evaluated, with the original fitness function, at random [69].

If the computational cost of the original fitness function is high then the individual based evolution control can be carried out in a selected number of generations [20].

With generation based evolution control the entire generation is tested, using the original fitness function, every n generations [120, 121] (Where n is a fixed number decided upon before the GA is started).

Adaptive Evolution Control As the accuracy of the approximation increases less evolution control needs to occur. With some approximation models it is possible to use the information gained from the evolution control to increase the accuracy of the model. A GA using such an approximation can use an adaptive evolution control that will adjust the amount of control needed based upon the current accuracy of the approximation model.

The trust region network [35] has been suggested as the basis for a method to implement adaptive evolution control [108]. An alternative framework for approximate model management has been suggested and applied to 2 dimensional aerodynamic design optimisation [70].

1.4 Hybrids

It is possible to combine GAs with a secondary method to create a hybrid GA (also referred to as a Memetic algorithm [107]). Hybrid GAs usually consist of a GA combined with either a local search (for a general problem solver) or a heuristic (for a more problem dependant solution) [137].

Hybrid GAs can provide a number of advantages over a standard GA.

Speed Quicker convergence to the optimum once the GA has located a promising area in search space.

Repair Replacing invalid individuals with similar valid individuals. This is very useful if the crossover operator used does not guarantee to produce a valid individual [112, 68].

GA functional enhancement The genetic operators used by a genetic algorithm may be enhanced or replaced with a secondary method. E.g. a neural network may act as a fitness estimator for the fitness function. [64].

1.4.1 Hybrid Architecture

Hybrid GAs can be classified by the way the GA uses the secondary method. Yen et al [151] provided a classification for Hybrid GAs which was enhanced by Sinha et al [137]. Their classification divides hybrid GAs into the following classes. (Note: the classes are not mutually exclusive so a hybrid GA may fit into more than one class, eg. a Postprocessor Pipelined Hybrid usually also fits into the Embedded Initialisation category.)

Pipelined Hybrids consist of two distinct sequential stages (the GA and the secondary method) A pipelined hybrid can be one of the following types.

- **Preprocessor:** The GA is used first to locate good locations in search space, these locations are then used to initialise the secondary method.

- **Postprocessor:** The secondary method is used to provide the initial population for the GA [39, 119].

- **Staged:** The GA and the secondary method are interleaved in a loop. First one method will run then the results of that will feed into the other method whose results feed back into the first. For example a GA may produce a new generation which then has a local search performed on each individual and the best one found for each individual searched is put back into the GA's new population to be selected for crossover and mutation [96, 39].

Asynchronous Hybrids The GA and the secondary method run in parallel and store solutions in a shared memory. As each search method deteriorates it is started again from the best solution in the shared memory [139, 38].

Hierarchical Hybrids have multiple levels of optimisation. For example Rogers [129] used a GA / linear regression hierarchical hybrid to evolve function approximation using splines. The GA evolved the basis function while the coefficients that produced the least error were discovered through linear regression.

Embedded Hybrids have the secondary method embedded within part of the GA itself. Embedded Hybrids can be broken down into one of the following subcategories

Initialisation is where the secondary method is used to generate the initial population for the GA [119, 39].

Fitness Evaluation is where the secondary method is used to evaluate the fitness of an individual. Neural Networks have been a popular secondary method to use for this [64].

Crossover in certain problems may result in invalid individuals being produced (e.g. The Travelling Salesman Problem). A secondary method may be used to generate valid individuals or to repair invalid individuals [149, 128].

Mutation may use a secondary method to generate new individuals in the neighbourhood of the existing individual [51].

Special Operators that use the secondary method may be created that the GA can use as well as or instead of the traditional genetic operators [138].

1.4.2 Secondary Methods

Many secondary methods have been used in hybrid GAs. The following list is not exhaustive but gives a good idea of how varied the area of hybrid GAs has become.

Local Search Methods usually operate on an individual solution. The local search moves from a location in search space to a neighbouring location with a higher fitness level. The local search continues moving from neighbour to better neighbour until it reaches a local optimum (i.e. there are no neighbours with a better fitness level) [137]. Local search methods used include

- Newton's Method [110]
- Steepest Descent [99]
- Broyden, Fletcher, Goldfard and Shanno's Method [19]
- Powell's Method [116]
- Conjugate Gradient Method [75]

Simulated Annealing is based on the concept of annealing molten metal [77]. Simulated Annealing works with a single solution and avoids getting trapped in local minima by accepting non-improving moves according to the probability set by the Metropolis criterion [100]. The Metropolis criterion is shown in Equation 1.1 where δf is the increase in the function value and T is the control parameter (equivalent to the temperature in the annealing scenario). As the temperature decreases over time so to does the probability of accepting a non-improving move.

$$p = exp\left(-\frac{\delta f}{T}\right) \tag{1.1}$$

Artificial Neural Networks are inspired by how the brain works. In an artificial neural network nodes (called neurons) are connected with synapses (directed connections). Every synapse has an assigned weight and every neuron has a transfer function (usually either a sigmoid, Heaviside or Gaussian function). The output of each neuron is calculated by the values of the input synapses (with their weightings taken into account) processed by the transfer function.

GAs have been combined with artificial neural networks to tackle many problems [132, 150]. Sinha et al. groups these hybrids into 5 categories [137].

Determination of an artificial neural network topology where the number of layers, neurons and the synapses between neurons are evolved by the GA[59, 61].

Training of an artificial neural network where the GA evolves the weights of the synapses between the neurons. This has been shown to produce better results than the traditional training method of error back-propagation [106].

Selection and generation of training data where the data set used to train the artificial neural network is evolved by the GA. The training set used has a direct effect on the ability of the artificial neural network to tackle the same problem for different sets of data [127].

Artificial neural networks for fitness evaluation replaces the fitness function in the GA with an artificial neural network that has been trained to model the problem being tackled [64].

Input feature selection uses a GA to evolve a smaller feature set for the artificial neural network and thus reduces the complexity of the resulting artificial neural network[115, 150].

Tabu Search is a discrete optimisation method that avoids getting trapped in local minima by not moving to locations searched recently [48, 49]. Each iteration the tabu search moves to the best solution available in the neighbourhood. Solutions that are encountered frequently or have only recently been encountered are stored in a tabu list and while in this list are not valid locations for the tabu search to move to. Locations stay in this list for a specified length of time (referred to as the tabu tenure).

Tabu search has been used in GA hybrids as a local search to improve each offspring [39]. Glover et al. suggested the use of a tabu search to provide a strategic oscillation in GAs [50]. The idea being that the GA's population consists of a combination of feasible and infeasible solutions because depending on the search space it can be easier to reach the global optimum through the infeasible locations in the search space.

Case-Based Reasoning is a method used to store previous solutions in a case book. These stored solutions are uses as a basis when tackling similar problems in the future. In a hybrid GA with case-based reasoning the GA's population is seeded with the individuals stored in the case book for similar problems [94, 119].

1.4.3 Lamarckian Evolution and Baldwinian Learning

Comparisons can be drawn between hybrid GAs and biological systems [6]. The GA part of the hybrid relates to the evolution of a species while the secondary method relates to the traits learnt by individuals of that species during their lifetime. Hinton et al. have shown that evolution can be guided indirectly by learnt traits of individuals [65]. The mechanism for this indirect guidance is referred to as the Baldwin [5] effect. For learnt traits to affect evolution in nature there has to be strong correlation between the traits learnt and the environment being evolved in. This is not a problem for hybrid GAs where the GA and the secondary method are normally solving the same problem.

There are two mechanisms that can be used to let the secondary method affect the evolution of the GA, Lamarckian[2] Evolutions and Baldwinian[3] Learning [65].

Lamarckian Evolution replaces an individual and its fitness after it has been used by the secondary method with the best result returned by the secondary method.

Baldwinian Learning replaces only the fitness of an individual after it has been used by the secondary method with the best fitness returned by the secondary method.

Lamarckian evolution is believed to disrupt the exploration capability of the GA and in some cases leads to convergence on local minima [146]. Orvish et al. suggest that Lamarckian evolution is only use once in every 20 trials to avoid these problems [112].

1.4.4 Secondary Method's Duration

An important question for hybrid GAs is how long to allow the secondary method to run for. Mathias et al. [96] argue that running the secondary method till convergence can have a detrimental effect on the diversity of the GA's population and can result in a large number of costly fitness function evaluations.

For certain classes of problems the fitness of an individual after a small change can be calculated a lot quicker than calculating it from scratch (e.g. the Row-Based VLSI Layout Problem [110]). Radcliffe et al. called these decomposable functions and argued that hybrid GAs are suitable for tackling these types of problems [118]. Research into the optimal duration for the secondary method of a hybrid GA has so far not produced any clear evidence about the effect on the performance [62, 87].

[2]Lamarckian refers to Lamarck[86] an early 19^{th} century biologist

[3]Baldwinian refers to Baldwin[5] a late 19^{th} century biologist

1.5 Summary

The process of evolution can be broken down into two distinct stages, selection and inheritance. Selection (be it natural selection or unnatural selection) is the process where the parents of the next generation are selected. Inheritance enables the next generation to keep some of the attributes their parents had. These combined with the chance of mutation enables a species to improve and adapt to its surroundings.

As GAs are based on the theory of evolution they too consist of selection and inheritance. The selection is provided by the use of a fitness function to decide how good an individual is and a selection operator to decide which individuals to use as parents for the next generation. The inheritance part for GAs is provided by the encoding method and the crossover operator which helps keep the attributes intact when passed from parents to children.

GAs have been used to tackle many problems including the following

- Calibrating combustion engines [78].
- Antenna design [91].
- Vehicle Routing [141].
- Evolving Shell sort Sequences [135].
- Congressional Redistricting [42].
- Strip Packing Problems [55].
- Optimisation of wireless systems [66].

Hybrid GAs can improve on the performance achieved by normal GAs. These consist of a GA combined with a secondary method. The types of secondary method and ways that they can be combined with a GA are numerous. The information gathered by the secondary method may be directly recoded back into the GA (Lamarckian Evolution) or may just be used to direct the GAs evolution (Baldwinian Learning).

Many GAs can take a long time to run [57, 89, 79, 60, 108, 134, 113, 70]. Approximation models have been used as fitness functions to improve the performance of these GAs. These approximation models sacrifice accuracy for a increase in the speed of the fitness testing. The rest of this Thesis will look at two new types of fitness functions capable of improving the performance of a GA without sacrificing the accuracy of the GA.

Chapter 2

Motivation

2.1 Introduction

This chapter gives an overview of the work that motivated the rest of the research described in this Thesis. The research in this chapter was undertaken for a project funded by Nortel Networks. The funding for the project was cancelled when Nortel Networks closed their site that was responsible for it. At that point the research was directed towards answering the question posed by this chapter.

Tests were carried out to assess the effect that packet sizes, DiffServ settings and protocols have on delays when transmitting packets across the Internet. This sort of information is needed for assessing the Quality of Service that can be provided between two hosts. Knowing how these settings affect the delay of packets will help to assess the length and quality of the data to be sent. The Quality of Service is mainly of interest when sending real time data, such as sound or video images, where delays over 200ms produce a noticeable effect on interactive services[117].

2.2 What affects the delay of a packet

There are four common types of delays on networks.

- Transmission Delays
- Propagation Delays
- Processing Delays
- Queueing Delays

2.2.1 Transmission Delay

The transmission delay is the time taken for a packet to be put onto the network by the interface. This is also known as Serialisation Delay. It can be calculated with the formula shown in Equation 2.1.

$$TransmissionDelay = \frac{PacketSize}{Bandwidth} \tag{2.1}$$

This delay is affected by the bandwidth of the networks and the number of intermediate nodes in the route the packet takes. If there are a lot of low bandwidth networks in the route then the transmission delay will be high, the more intermediate nodes there are the higher the transmission delay will be. The transmission delay usually contributes a negligible delay to the overall delay of a packet [27].

2.2.2 Propagation Delay

The propagation delay is the time taken for the packet to travel the length of the network. It can be calculated using the formula shown in Equation 2.2.

$$PropagationDelay = \frac{Distance}{SpeedOfLight} \tag{2.2}$$

This delay is affected by the distance travelled by the packet. The more direct a route the lower this delay will be. For networking purposes the Speed Of Light is counted as $2 * 10^8$ instead of $3 * 10^8$. This is due to the fact the signal is travelling through a physical medium and not through a vacuum [27].

2.2.3 Processing Delay

The processing delay is the time taken for a network device to examine a packet and decide what to do with it. This delay depends on the device processing the packet. The more nodes in a route, the more processing delay that will be included in the overall delay [27].

2.2.4 Queueing Delay

The queueing delay is the delay caused by the packet sitting in a queue waiting to be processed by a network device. If the queue is full when the packet arrives, it will get dropped. This delay is the least consistent of the delays, as it depends on the amount of traffic on the networks. This is where the majority of variation in delay times come from [27].

2.3 Experiments

The following experiments were carried out to assess the amount of control a host has over the way packets travel to their destination over the Internet.

- Varying Packet Size
- Varying DiffServ Priority
- Varying Protocol (TCP vs UDP)

2.4 Equipment

The experiments were carried out using a Sun Sparc 4 workstation connected to the Internet through a 10Mb/s switched Ethernet connection. The test scripts were written in Python (Version 1.5.2) and used Traceroute (Version 6.0 Gold) and TCPtraceroute (Version 1.2). As delays from these programs are the round trip times, the results were halved to represent the estimated delay for the journey of the packet to its destination. All experiments only had one packet on the network at a time so there would be no interference from the test script itself.

Two versions of the Traceroute program were used, the standard Traceroute program uses the UDP protocol while TCPtraceroute uses the TCP protocol. Traceroute detects the path that is used to connect two hosts together across an IP network (in this case the Internet). It works by sending multiple packets to the destination host, the first packet has a Time To Live of 1 and at the first hop in the route the packet times out and the sending host is informed when and where the packet has timed out. The second packet has a Time To Live of 2 and at the second hop in the route the packet times out and the sending host is informed of when and where the packet has timed out. The Traceroute program can use this information to display the route taken and how long each step in the trip took. It is important to note that as the measurements have to be taken by sending packets and waiting for the return packet Traceroute will actually have an effect on the traffic of the network, though this effect will be small.

2.4.1 Destinations used in the tests

Table 2.1 shows a list of destinations used for the experiments. The IP address and host name is given along with their physical location and the maximum transmission unit (MTU) size. The MTU's were obtained using Traceroute.

Table 2.1: Destinations used for the experiments

Destination	Host Name	Location	MTU
216.115.108.245	img5.yahoo.com	San Jose, USA	1492
194.135.30.46	weblist.ru	Russia	1492
18.181.0.31	DANDELION-PATCH.MIT.EDU	MIT	1492
195.162.250.2	webserver1.absolute-sports.de	Germany	1492
202.33.28.186	ns.square.co.jp	Japan	1492
212.219.56.146	chandra.mirror.ac.uk	UK	1492
217.12.6.16	dial1.lng.yahoo.com	London, UK	1492
217.12.6.17	dial2.lng.yahoo.com	London, UK	1492

These destinations were chosen based upon their locations. The first 5 destinations enable the packet configurations to be assessed over different physical networks and determine if the settings of the packets gave consistent results when travelling over different routes. The last 3 destinations enable the packet configurations to be assessed over routes that share a majority of underlying networks.

2.5 Varying Packet Size

This test was to examine the effect on delays of changing the packet size. Theoretically, the smaller the packet size the smaller the delay for an individual packet. However there would be more packets required to transmit the same amount of data. For messages larger than a single packet, the larger packet sizes usually result in a more efficient data transmission as fewer packets are being sent. However if the data being transmitted is real time data then there would be an additional delay for each packet for the time taken to acquire the data to be sent, hence the smaller the packet the lower the acquisition delay for each packet.

2.5.1 Algorithm Used

The algorithm used in the test script is shown in Algorithm 1. The packet size includes the header data.

The algorithm waits for each instance of traceroute to complete before starting the next which resulted in approximately 15 minutes between each reading for each packet size.

Algorithm 1 Algorithm to test the effect on delay by varying packets size

```
$D$ = img5.yahoo.com (216.115.108.245)
loop
  for $x = 40$ to 1500 do
    call traceroute with $D$ and packet size of x
    log date, time and delay
  end for
end loop
```

2.5.2 Results

At the start of the graph shown in Figure 2.1 the small and medium size packets have a drop in their delay time, this is not present to the same degree for the large size packets. Otherwise the delay for all of the three packet sizes shown is very similar, with the 1500B packets taking longer than the 750B packets. It is interesting to note that the 750B packets do not always take longer than the 40B packets. The 1500B packet will be fragmented as it is larger than the MTU for the route, this helps explain why it does not show the same reduction in the delay as the other packet sizes.

This graph demonstrates why it is important to test the settings over a long period of time. If the decision on the best packet size to be used was made at the start of the test period where there is a large trough (probably caused by a lower than normal load on the intervening networks in the route) then the packet size of 40B would be chosen, which at times performs worse than 750B packets.

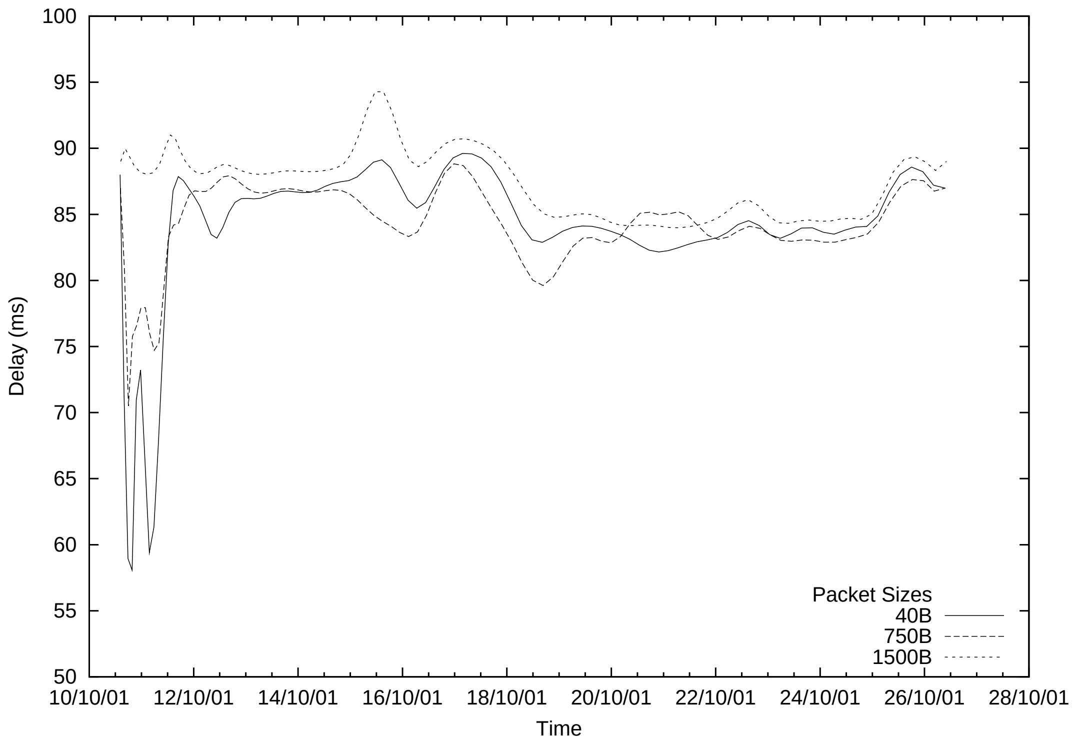

Figure 2.1: Delay of packets to img5.yahoo.com (216.115.108.245), San Jose, between the 10th October and the 28th October 2001. Packet sizes of 40B, 750B and 1500B shown

2.5.3 Summary

The majority of the time there is very little difference between the delay on the packet sizes, 2ms or 3ms and at the most 10ms (with the exception of the 1500B packet in Figure 2.1). This indicates that the majority of delay comes from propagation delay, processing delays and queueing delays as these are the only ones left after eliminating packet size (Transmission Delay). The only times that major differences occur is where the delay time for the 1500B packets in Figure 2.1 do not drop when the other packets do. This is best explained by the 1500B packet being too big for the networks that it needs to travel through. As the MTU is 1492B for the route used, the packet would have to be fragmented to pass through. This would result in a delay for the multiple packets to be transmitted and recombined. While only one site has been used as the destination in this experiment the results of the experiment show that changes in the packet size do not always affect the delay as would be expected.

2.6 Priority using DiffServ

This test was designed to see if there is a difference in the delay on packets depending on the settings of the DiffServ bits in the packet header.

2.6.1 Algorithm Used

The algorithm used to test the effect that DiffServ has on delays is shown in Algorithm 2.

Algorithm 2 Algorithm to test the effect DiffServ settings have on delays

```
loop
  for $D$ in destinations do
    for $P = 0$ to 255 do
      call traceroute with $D$ and priority setting of $P$
      log date, time, priority and delay
    end for
  end for
end loop
```

2.6.2 Mapping from Traceroute to DiffServ

The DiffServ bits are an octet passed to traceroute, the value is between 0 and 255. This is mapped to the equivalent DiffServ settings using the following method. Bits 0,1,2 are the class. Bits 3,4,5 are the priority within each class. Bits 6 and 7 are not used in the DiffServ protocol at the time of writing.

2.6.3 Results

The following graphs all show the propagation delay expected for a packet travelling from the source host to the destination shown at the bottom of the graph. The transmission delay is negligible and makes very little difference to the result. The difference between the propagation delay and the actual delay comes from queueing delays and processing delays.

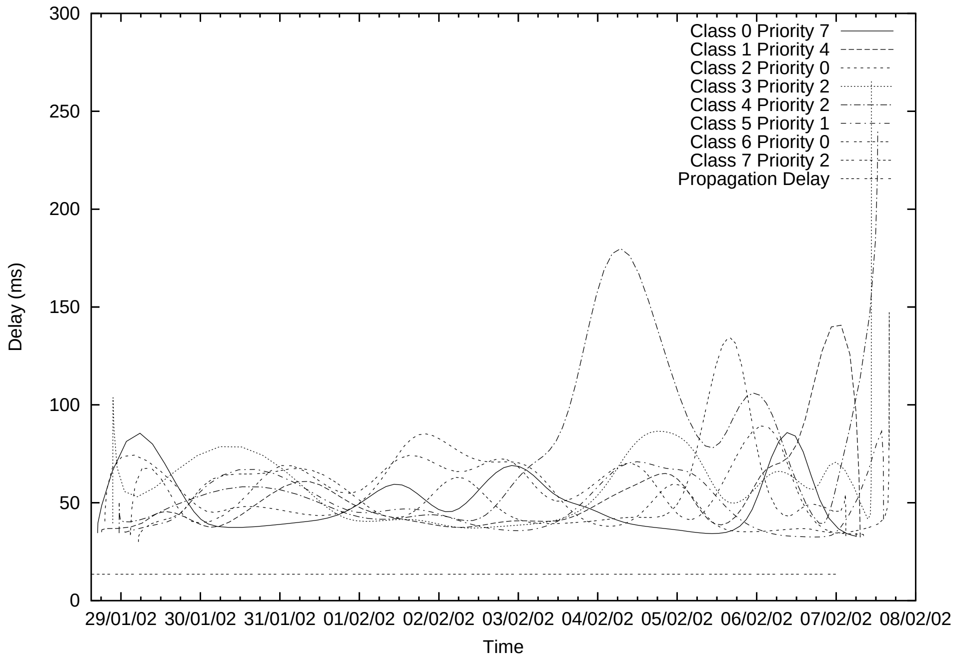

Figure 2.2: The best results for each of the 8 different classes between 19th January and 7th February 2002 to weblist.ru (194.135.30.46), Russia

The graph in Figure 2.2 shows the best of the priorities for each class going to weblist.ru (194.135.30.46), Russia. At any time there is a class and priority setting which has a delay below 50ms but some of them only stay below this level for some of the time and none of them stay below it all the time.

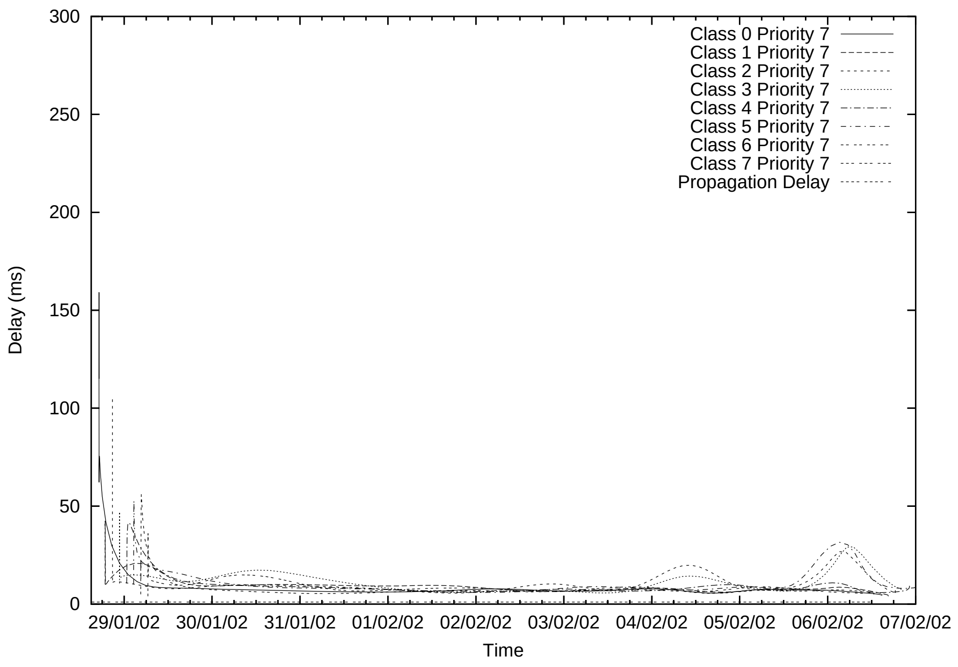

Figure 2.3: Delay of packets to dial1.lng.yahoo.com (217.12.6.16), London, showing priority 7 for all 8 different classes

The graph in Figure 2.3 shows that during the stable period only class 2, 3 and 4 stand out from the others, by having peaks which are large when compared to the other classes, while the rest of the classes only suffer a small delay. The spikes that appears on the graphs shown in figures 2.3 and 2.4 all appear around the 29^{th} January 2002. The spikes were most likely caused by a problem on a network in the routes involved (either heavy use or equipment failure). It does however illustrate the need to assess the packets configuration over at least two weeks.

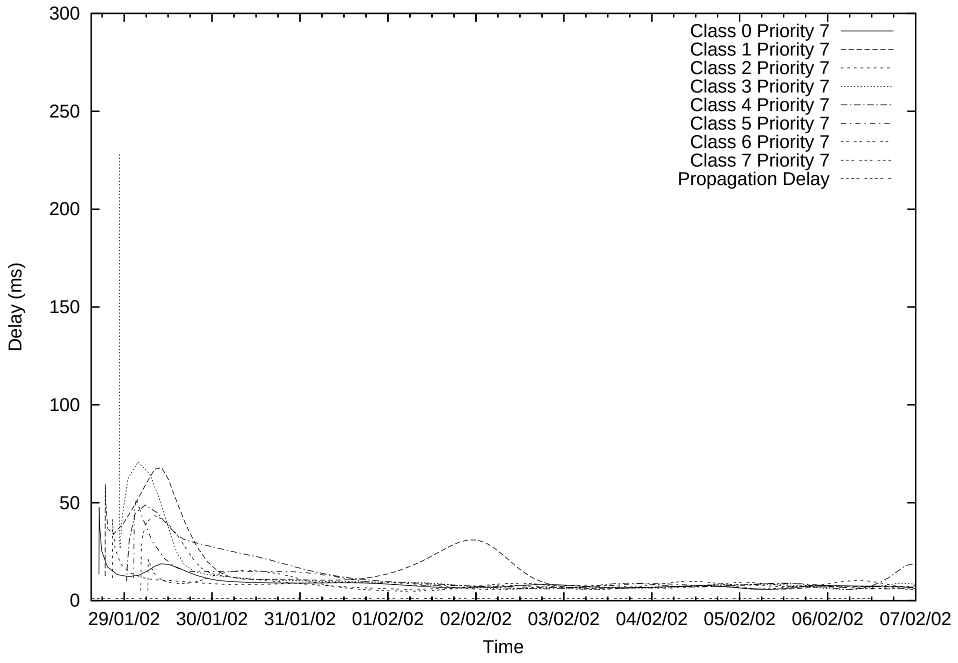

Figure 2.4: Delay of packets to dial2.lng.yahoo.com (217.12.6.17), London, showing priority 7 for all 8 different classes

Once the delays start to settle down after the first part of the graph, shown in Figure 2.4, there is very little difference between the classes. Class 1 and Class 4 are the only ones to stand out as they have peaks in the later part of the graph.

2.6.4 Summary

The delays to weblist.ru (194.135.30.46), Russia, seem to be fairly unstable as no class gives a consistent delay.

The delays to dial1.lng.yahoo.com (217.12.6.16), London, get the better results using class 0, 4, 5, 6 and 7. Class 7 gets the best result of all, for both consistency and actual delay times.

The delays to dial2.lng.yahoo.com (217.12.6.17), London, get the better results using class 0,1,2,3,6 and 7. Class 6 gets the best result for both consistency and actual delay times. The results for classes 1,2,3 are different to dial1.lng.yahoo.com (217.12.6.16) which is surprising as both hosts are on the same network. Class 0 is the only class which gives a consistently good result. It is interesting to note that Class 0 is the default setting for the class and so leaving the default settings for packets is probably better than guessing at a DiffServ setting.

Changing the DiffServ settings on the packets certainly makes a difference to the delay and in some cases it can make a large difference. More testing on this needs to be done to understand why the results for the two destinations on the same network gave very different answers in some cases and to see if good settings for a destination vary with time, or if good settings vary as the destination varies. It is interesting to note that there seems to be almost always a setting which gives acceptably low delays to any given destination.

2.7 Different Protocols

This test was designed to see if there is a difference in delay between sending packets using different protocols. TCP and UDP were compared to see if they have different times in their delays.

2.7.1 Algorithm Used

The algorithm used to test the effect that the use of TCP and UDP has on the delay on packets is shown in Algorithm 3.

Algorithm 3 Algorithm to test the effect TCP and UDP have on delays

```
loop
  for $D$ in destinations do
    call traceroute with $D$
    log date, time, protocol (UDP) and delay
    call TCPtraceroute with $D$
    log date, time, protocol (TCP) and delay
  end for
end loop
```

2.7.2 Results

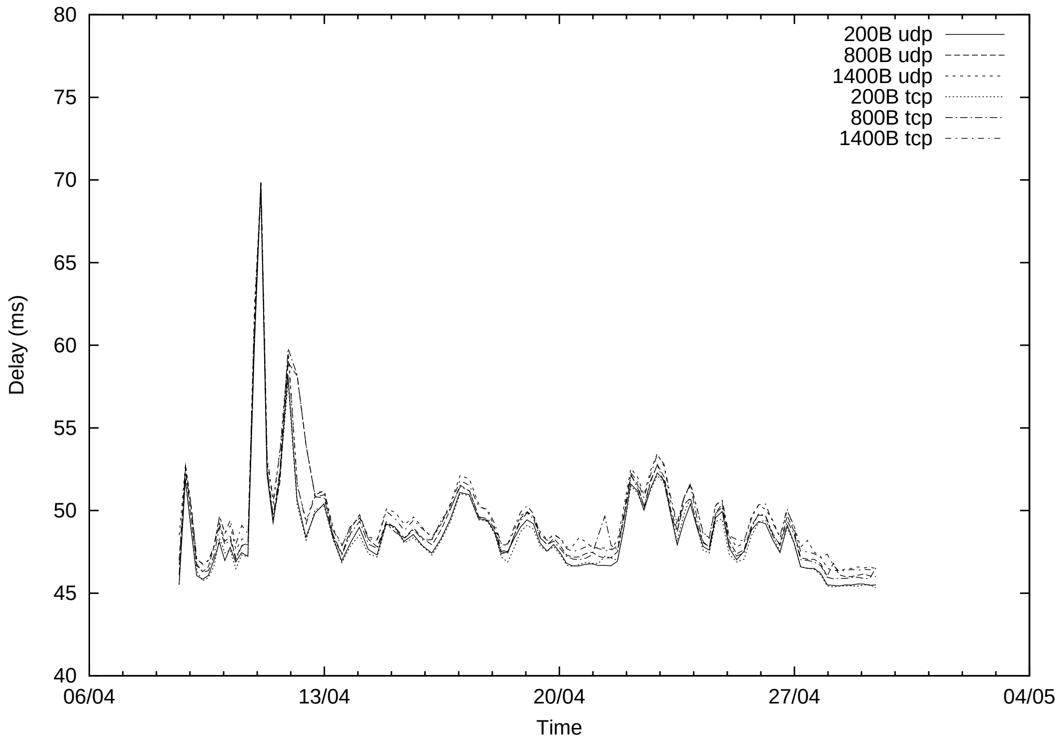

Figure 2.5: UDP and TCP times to DANDELION-PATCH.MIT.EDU (18.181.0.31), East Coast USA.

The graph in Figure 2.5 shows both the protocols have similar delays, neither one stands out as being better.

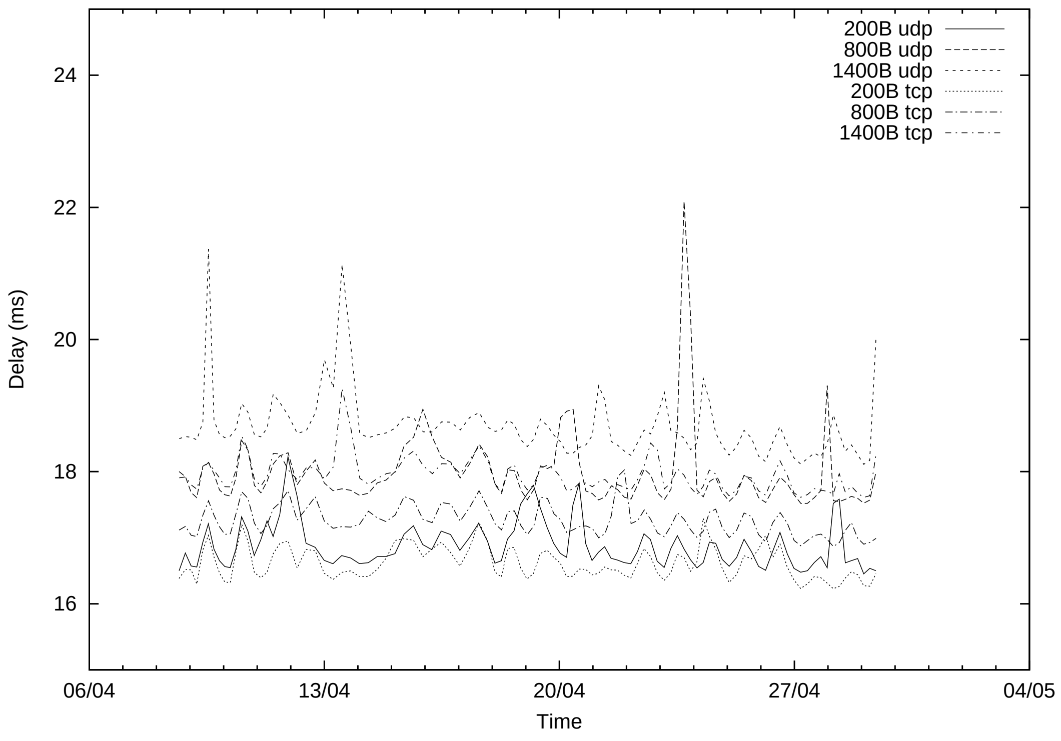

Figure 2.6: UDP and TCP times to webserver1.absolute-sports.de (195.162.250.2), Germany.

The graph in Figure 2.6 shows that again the sizes are close together, it is interesting to note that the lines for the TCP protocol have delays slightly below those of their UDP counterparts.

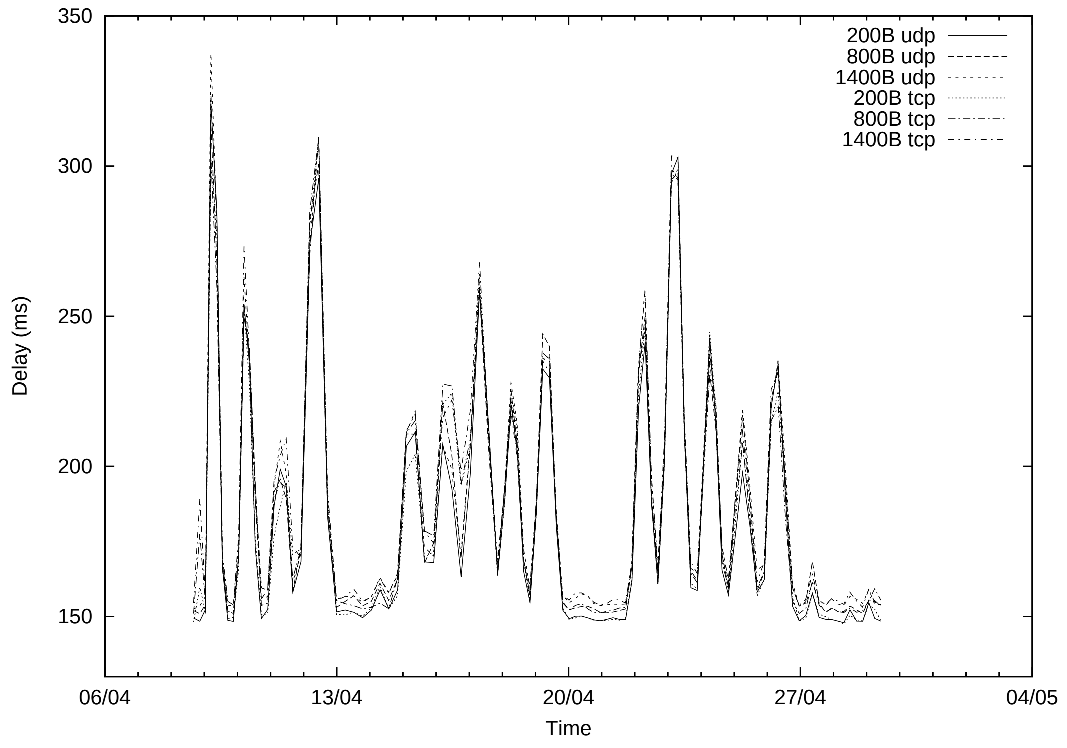

Figure 2.7: UDP and TCP times to ns.square.co.jp (202.33.28.186), Japan.

The graph in Figure 2.7 shows the delays for the different protocols are similar. Every now and again the TCP delays are slightly better than the delays for the UDP.

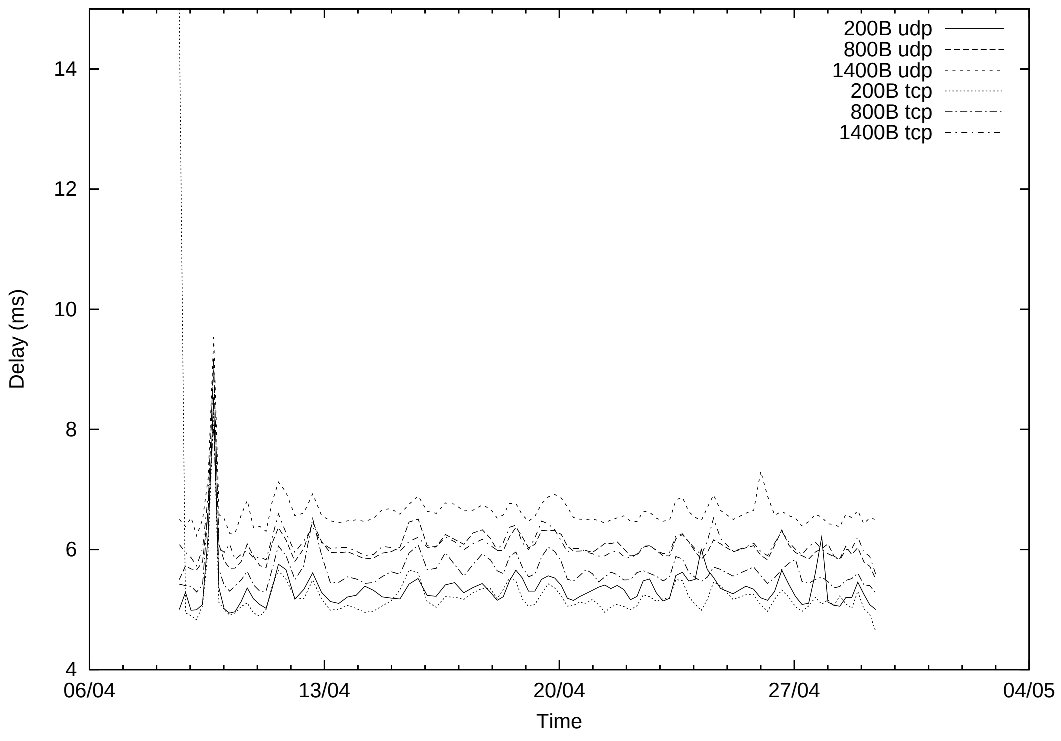

Figure 2.8: UDP and TCP times to chandra.mirror.ac.uk (212.219.56.146), UK.

The graph in Figure 2.8 shows that again the TCP packets seem to do marginally better than their UDP counterparts.

2.7.3 Summary

The delays on the TCP and UDP packets are very similar and are usually within 10ms of each other. This would indicate that using a different protocol does not help to avoid most of the delays, the only effect on delay would be a difference in the processing as it might be slightly quicker to process one type of packet than another.

It is interesting to note that while it is usually slightly better to use TCP there is no indication whether this affects the time taken to send large amounts of data. Also the TCP might have been affected by the fact that a UDP packet had just been sent to the same destination previously. If the difference in the delays had been greater then it would have been beneficial to rerun the test with the order reversed (TCP then UDP). As the difference between the protocol was small the suggested benefits did not warrant running the test with the order reversed.

2.8 Optimising packet configuration with a Genetic Algorithm

This Section looks at how a standard GA could be applied to the problem of optimising the configuration of a packet for a specific route.

2.8.1 What needs optimising?

The results from the previous Sections in this Chapter suggest that the DiffServ settings have the greatest effect on the delay of a packet, so the GA should optimise those settings. The GA will also need to optimise the packet size settings as well as the DiffServ Settings if the networks in the route do not support the method described in RFC 1191 [104] for discovering the MTU.

2.8.2 Genome Length

Assuming that a binary encoding is used the genome will need 6 bits to encode the DiffServ parameter and 11 bits to encode the packet length to be used. This results in a genome with a total length of 17 bits.

2.8.3 Fitness Testing

Due to the Internet being a noisy environment the fitness test for each individual will need to take place over a period of at least a week. Reducing the length of the fitness test will increase the risk that the individuals' fitness levels would not be a true indication of its ability to solve the problem at other periods. Increasing the period to longer than a week would be preferable as this will help reduce the effect of the noisy environment being evolved in. Two weeks is a reasonable length of time to test an individual's fitness over and so will be used as the length of the fitness test in the calculations in the next section.

2.8.4 Running Time Span

With a genome length of 17 bits there are 131072 possible combinations. To brute force all possible individuals would take over 5041 years to complete. A GA using a population size of 60 that was run for 400 generations would only take 923 years. The GA takes less time to run than the brute force option but 923 years is still far too long to be of any use as a solution to this problem.

A possible optimisation in the fitness testing of individuals is that a number of individuals could be checked in the same two week period by cycling through them (Note that this would reduce the number of times the route would be checked for each individual and as such would increase the effect that the noisy environment would have on the individuals fitness level). Assuming that 60 individuals could be tested in the same two week period the length of time it would take to run the GA for 400 generations would be over 15 years.

Increasing the number of individuals that can be tested in the same 2 week period beyond 60 will not reduce the length of time it takes the GA to run for 400 generations but would increase the size of the population that the GA can use without increasing the time taken to test a generation.

Using a GA instead of a brute force method reduces the running time span greatly but it does not reduce the time enough to be an acceptable method to solve this problem.

2.9 Overall Conclusions

Of all the tests carried out the greatest differences were due to the DiffServ settings, however these were not consistent across all tests.

The packet size plays a small part in the delay, but it seems best to use the largest packet size as possible as this reduces the number of packets and hence the number of headers that need sending.

The DiffServ bits give the greatest hope for some control over the delay on packets, unfortunately there does not seem to be a specific setting which performs well to all destinations. There may be a setting which performs well to a specific host and this might stay constant, though these settings would need to be discovered for each host the packet is being sent to. More research needs to be carried out into this to determine exactly what the effect of the DiffServ bits have on a packet's delay and in what circumstances.

The TCP and UDP test show that TCP packets are usually marginally faster than packets being sent using UDP.

To get a long term view of the performance of a specific configuration of a packet, the configuration would have to be tested for at least two weeks due to the constantly changing nature of the Internet. After completing these tests and the length of the fitness test that would be needed if a GA was applied to the problem became apparent the question was posed "How could genetic algorithms be improved to decrease the length of time needed to evolve an acceptable solution?". The rest of this Thesis introduces and investigates two methods that improve the performance and efficiency of GAs which could be applied to this problem to answer the question posed.

Chapter 3

Population Sizes and Level of Elitism

3.1 Introduction

This chapter investigates the effect that population size and the level of elitism has on the efficiency and performance of GAs. Section 3.3 describes the test problems used to analyse the effects that population sizes and the level of elitism have on a GA tackling problems with different degrees of difficulty. Sections 3.5.1 and 3.5.2 examine the effects that population sizes have on the performance of a GA. Section 3.5.4 examines the effects that population sizes have on the efficiency of a GA. Finally section 3.5.5 examines the effects that the level of elitism has on a GA.

3.2 Efficiency and Performance

The definition of the efficiency of a GA used in this Thesis is shown in Equation 3.1. This measure of efficiency for a GA is useful when assessing how much time a GA has used up calculating the fitness of known locations in search space. As the efficiency of the GA improves the number of locations in search space that have been searched also improves or the time taken to search those locations reduced.

$$Efficiency = \frac{Locations\ In\ Search\ Space\ Searched}{Time} \tag{3.1}$$

The definition of the performance of a GA used in this Thesis is shown in Equation 3.2. With this performance measure it will be possible to see if a change to a GA increases or decreases the level of fitness it can achieve in a specific length of time.

$$Performance = \frac{Fitness}{Time} \tag{3.2}$$

With respect to fitness functions that take a long time to compute, improving the performance of a GA is more important than improving the efficiency of the GA. The overall aim of a GA is to gain as high a level of fitness as possible in the available time. Improving the GAs efficiency by increasing the area of search space that has been searched does not guarantee an improvement in the performance of the GA. Reducing the length of time that a GA takes to get to the same level of fitness will improve both the performance and efficiency of the GA.

3.3 Test Problems

The following are the three basic test problems that have been used to examine the performance of GAs :

- One Max Problem [51, 40]
- Deceptive Trap Functions [51]
- GA Hard Problem

These three problems present three different degrees of difficulty for a GA. The One Max problem is an easy problem for GAs to tackle, while the Deceptive Trap problem is more difficult for a GA to solve. The GA Hard problem is even harder than the Deceptive Trap problem for a GA to solve[1], due the excessive amounts of local minima. Each GA setting will be tested on all three problems to examine how the difficulty of a problem affects the efficiency and performance of the GA.

3.3.1 One Max Problem

The One Max Problem is an easy problem for GAs to tackle, due to the smooth hill it presents. This makes it very useful for examining the effects of changes made to a GA.

The One Max Problem consists of evolving a bit string of a specific length, the more ones in the bit string the higher the fitness of that bit string. The standard fitness

[1]The GA Hard Problem is configurable in its level of difficulty, the parameters used throughout the work in this Thesis are shown in Section 3.3.3

function for the one max problem is defined as the following, where l is the length of the bit string and x is the bit string being tested.

$$f(x) = \sum_{i=0}^{l} x_i$$

The length of the bit string (l) used for the One Max Problem was 1000 bits, this gives a maximum achievable fitness level of 1000. As the One Max Problem is an easy problem for GAs to tackle a large size of bit string was selected to stop the GA solving the problem in the early generations.

3.3.2 Deceptive Trap Functions

Deceptive Trap Functions are a difficult problem for GAs to tackle[23, 54]. This makes them a good function to use as a fitness function to examine the performance of a GA, especially for examining the effects population sizes have on a GA's performance[23, 54]. An example of a 4 bit trap function is shown in Figure 3.1. It is similar to a One Max Problem except that the greatest available score is for a bit string consisting of all 0s.

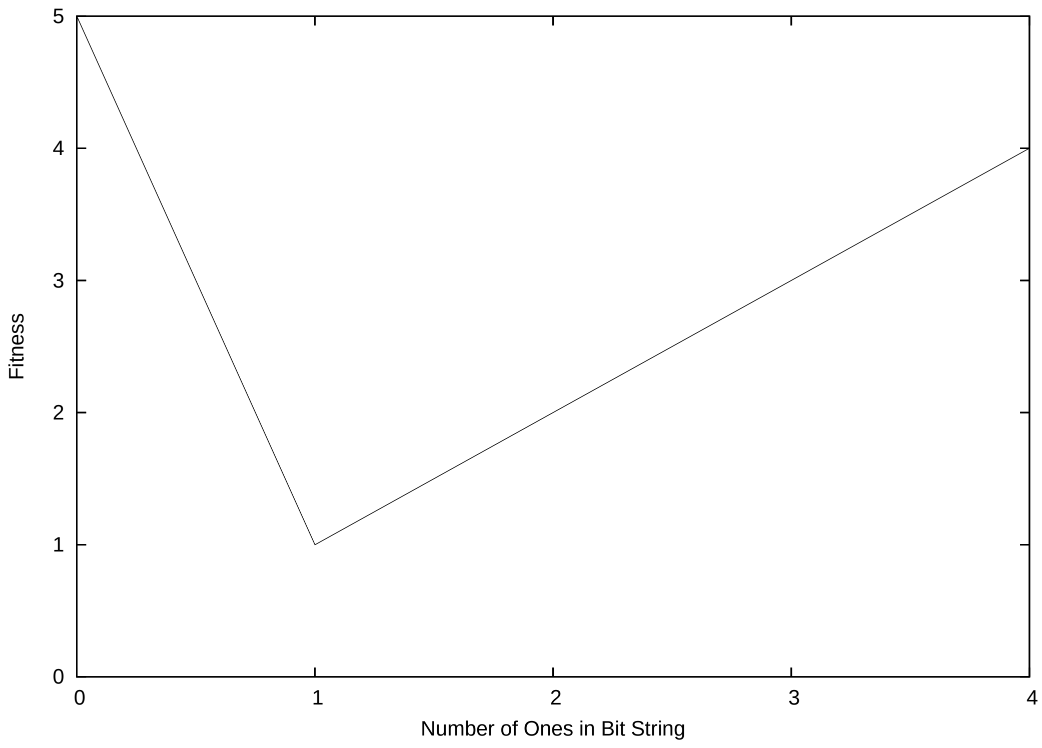

Figure 3.1: A simple 4 bit deceptive trap function

To make the problem hard enough to test the GA twenty of these problems were concatenated together, which gives a genome length of 80 bits. So the bit string is split up into 20 blocks, with each block 4 bits in length. This total scored by the 20 blocks is the fitness of the individual being tested. The maximum score for this test is a fitness of 100.

The big difference between the One Max Problem and the Deceptive Trap Functions is that the GA tackling the One Max Problem is always pointed in the correct search direction by the individuals' fitness levels, while the GA tackling the Deceptive Trap Functions is not always pointed in the correct search direction by the individuals' fitness levels.

3.3.3 GA Hard Problem

The GA Hard Problem is a function that has been designed at Loughborough University to be configurable in the level of difficulty it poses to a GA. An individual is mapped into an array, i, which consists of a number of eight bit integers. The array is then used to calculate the distance from a point C, coordinates c_k, using Equation 3.3. The result of Equation 3.3 is then used in Equation 3.4 to calculate a fitness value for the individual. An example of the surface produced by this Equation is shown in Figure 3.2.

$$dist = \sqrt{\sum_{k=1}^{dim} (i_k - c_k)^2} \tag{3.3}$$

$$Fitness = 10 * \cos\left(2 * \pi * \frac{dist}{rad} * amp * \left(2^{-\frac{dist}{ahl}}\right) + height * \left(2^{-\frac{dist}{hhl}}\right)\right) \tag{3.4}$$

The parameters used in the equation were selected by a process of trial and error. The trials consisted of plotting the graphs of the hills produced by the parameters being tested and then testing those that produced a hill with a large number of local minima on a standard GA. The parameters were selected so that the GA could improve in the early generations but as the GA approached the maximum it found it harder to improve.

The Number of dimensions (dim) parameter adjusts the size of the genome needed to tackle the equation. Increasing this value increases the size of the genome needed.

The Radius (*rad*) parameter adjusts the frequency of the troughs. Increasing this value decreases the frequency of the troughs.

The Height parameter adjust the base height of the hill. The higher the value the higher the hill will be based. This parameter is useful for some systems that have problems with negative fitness values as it allows the base height of the hill to be raised to avoid them.

The Amplitude (*amp*) parameter adjusts the base depth of the troughs. Increasing this value increases the maximum depth of the troughs.

The Height Half Life (*hhl*) parameter adjusts how quickly the height of the hill approaches the base of the hill. Decreasing this value increases the rate that the hill's height approaches the base.

The Amplitude Half Life (*ahl*) adjusts the decay of the depth of the troughs on the hill. Increasing this value reduces the rate decay.

The parameters chosen to be used in the tests are shown in Table 3.1. These parameters were selected by a process of trial and error as described earlier in this Section.

Parameter	Description	Default Value
dim	Number of Dimensions	5
Rad	Radius	1
height	Height	10.0
amp	Amplitude	2.5
hhl	Height Half Life	5
ahl	Amplitude Half Life	200

Table 3.1: Default Parameters for the GA Hard Problem

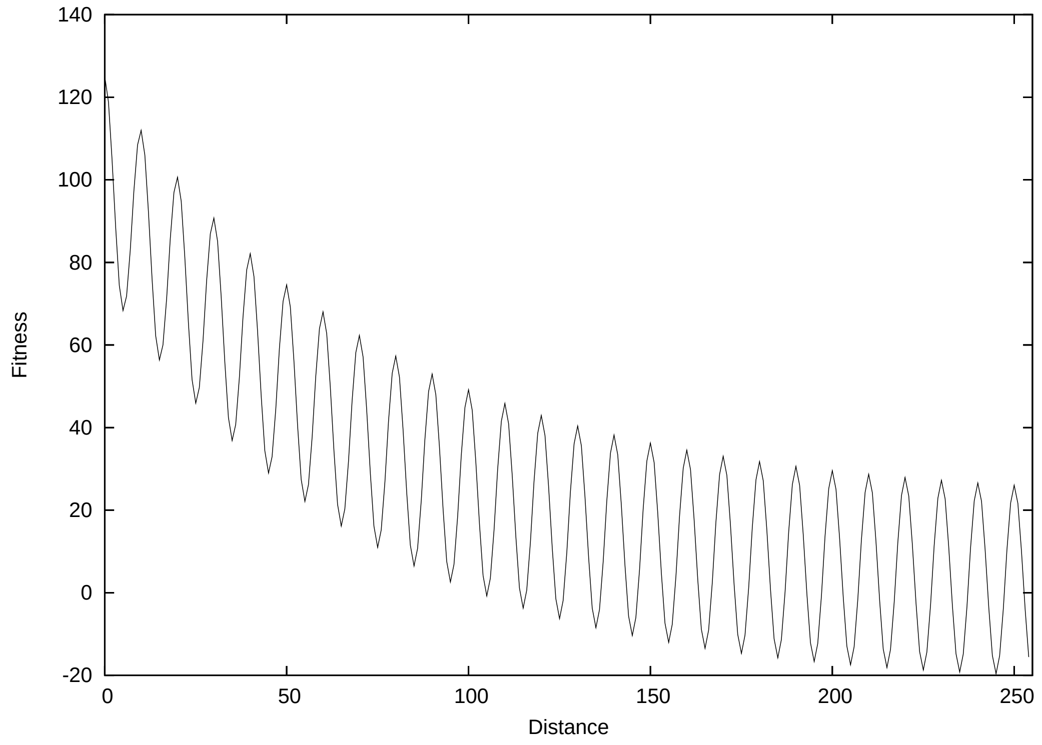

Figure 3.2: An example of the type of hill produced by the GA Hard Problem using the default parameters.

As can be seen in Figure 3.2 the GA Hard Problem, using the default parameters, has so many local minima that it is very hard for a GA to evolve without getting trapped in one. The GA Hard Problem was configured to be the hardest of the three test problems. The highest achievable level of fitness for the GA Hard Problem is 125.

3.4 Experiments

Four sets of experiments were undertaken to assess the following

- Population Sizes and their Effects on Performance
- Population Sizes and their Effects on Performance Using a Different Model of Elitism
- Population Sizes and their Effects on Efficiency
- Elitism Levels and their Effects on Performance

The effect of elitism levels on the efficiency of a GA has not been included in this set of experiments as the more individuals carried forward to the next generation reduces the number of new individuals that can be present in the next generation. Thus increasing the level of elitism reduces the efficiency of the GA. As this is known an experiment is not needed to assess it.

3.4.1 Elitism Levels Used

The GAs used in the three following sets of experiments are based on De Jong's Simple Elitist Genetic Algorithm[34] which carries the best individual forward to the next generation.

- Population Sizes and their Effects on Performance
- Population Sizes and their Effects on Efficiency
- Elitism Levels and their Effects on Performance

The effect of increasing the number of individuals carried forward has on the GA using a population size of 6 is investigated in the "Elitism Levels and their Effects on Performance" experiment (See Sections 3.4.5 and 3.5.5).

The "Population Sizes and their Effects on Performance Using a Different Model of Elitism" set of experiments investigates the effects population size has on a GA which is carrying a percentage of the best individuals forward to the next generation.

3.4.2 Population Sizes and their Effects on Performance

A comparison of a GA using three different population sizes (6, 60 and 600) was performed against the three test problems. The number of generations was inversely proportional to the number of individuals in the population, so each run took approximately the same length of time. The GA was tested on all three of the basic test problems to see how the difficulty of the problem affected the performance of the GA. On all three problems the GA used elitism (at a level of 1, i.e. only the best individual was carried forwards to the new population). The GA used a mutation rate of $\frac{1}{n}$ and single point crossover.

For all three problems the GA was run 50 times with a population size of 6, then run another 50 times with a population size of 60 and then run another 50 times with a population size of 600. Each population size used the same set of 50 seeds for the pseudo random number generator. The GA with a population size of 6 was run for 4000 generations, the GA with a population size of 60 was run for 400 generations and the GA with a population size of 600 was run for 40 generations. This resulted in all three population sizes taking approximately the same length of time to run.

3.4.3 Population Sizes and their Effects on Performance Using a Different Model of Elitism

The second set of experiments assessed the effect that using a different model of elitism would have on the performance of a GA.

A GA was configured to carry the best $\frac{1}{6}$ of the population forward to the next generation. A comparison of a GA using three different population sizes (6, 60 and 600) was performed against the three test problems. The number of generations was inversely proportional to the number of individuals in the population, so each run took approximately the same length of time. All three of the basic test problems were tackled by the GAs to assess how the difficulty of the problem affected the performance of the GA. The GA used a mutation rate of $\frac{1}{n}$ and single point crossover.

3.4.4 Population Sizes and their Effects on Efficiency

The third set of experiments assessed the effect that population size has on the efficiency of the GA. A GA was run 50 times using three different population sizes (6, 60 and 600) against each of the test problems. The number of generations was inversely proportional to the number of individuals in the population, so each run took approximately the same length of time. The number of locations in search space searched was recorded as an indication of how efficient the GA was with each population size. The GA was tested on all three of the basic test problems to see how the difficulty of the problem affected the efficiency of the GA. On all three problems the GA used elitism (at a level of 1). The GA used single point crossover and had a mutation rate of $\frac{1}{n}$ (where n is the length of the genome). The mutation rate of $\frac{1}{n}$ has been selected based on the results of Ochoa [111].

For all three problems the GA was run 50 times with each of the three population sizes (6. 60 and 600). Each population size used the same set of 50 seeds for the pseudo random number generator. The GA with a population size of 6 was run for 4000 generations, the GA with a population size of 60 was run for 400 generations and the GA with a population size of 600 was run for 40 generations. This resulted in all three population sizes taking approximately the same length of time to run.

3.4.5 Elitism Levels and their Effects on Performance

The fourth set of experiments compared the effects that increasing the number of individuals carried forward had on the GA. For each level of elitism the GA was run against the three test problems. The levels of elitism used were

No individuals carried forward,

One individual carried forward (An elitism level of 1),

Two individuals carried forward (An elitism level of 2),

Three individuals carried forward (An elitism level of 3)

On all three problems the GA had a population size of 6, used a mutation rate of $\frac{1}{n}$ and single point crossover. The population size was chosen based on the results of the first three experiments (see Sections 3.4.2 to 3.4.4 and 3.5.1 – 3.5.4).

For all three problems the GA was run 50 times with no elitism, then run 50 times with an elitism level of one, then run 50 times with an elitism level of 2 and then run 50 times with an elitism level of 3. Each level of elitism and the run with no elitism used the same set of 50 seeds for the pseudo random number generator.

3.5 Results

The following sections (3.5.1 – 3.5.5) contain a summary of the results of the experiments. Appendix B contains a full set of results. The average fitness levels shown in the graphs in the rest of this chapter are the average of the best fitness level achieved for each of the 50 runs.

3.5.1 Population Sizes and their Effects on Performance

This section shows and discusses the results of the experiments examining the effects of population sizes on the performance of the GA.

One Max Problem

The following graphs show the results of the runs of the GA, with the varying population sizes, against the One Max Problem.

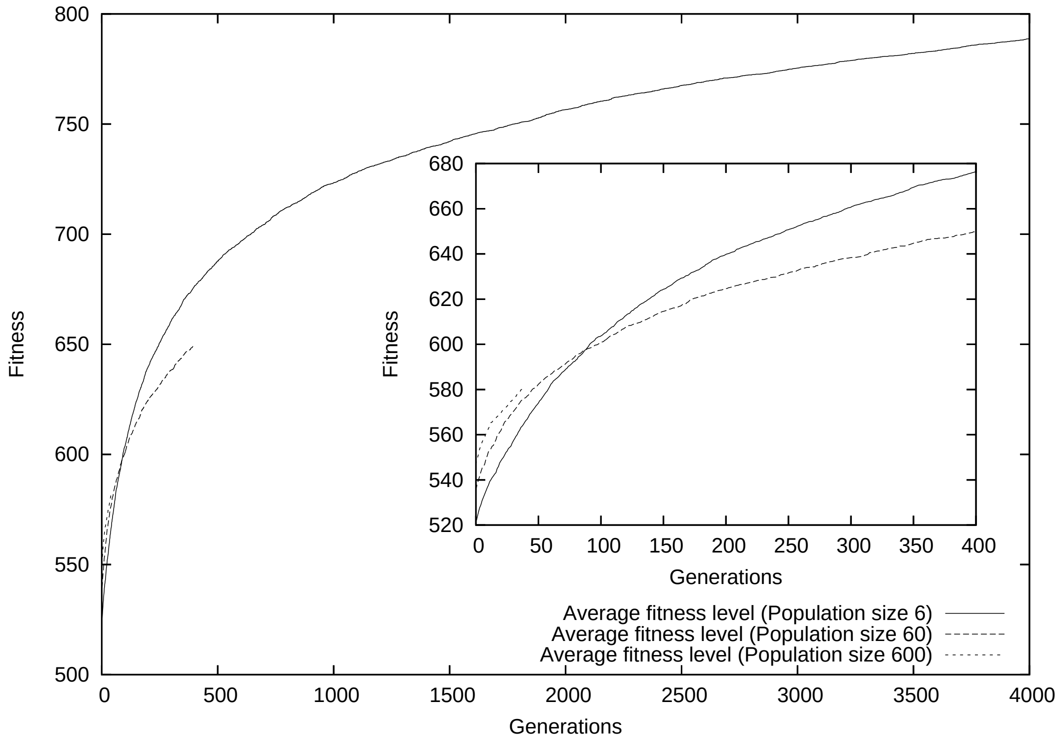

Figure 3.3: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against generations

The graph in Figure 3.3 shows the average best fitness of all three population sizes. They are shown against generations to show the difference in the number of generations the different population sizes can evolve for in the same length of time. While the higher population sizes do marginally better to start with, the population size of 6 manages to improve its fitness to a higher level than the other population sizes with the extra generations it can achieve in the time available to it. It is interesting to note that the population size of 6 performs better than the population size of 60 within the same number of generations, this is discussed later.

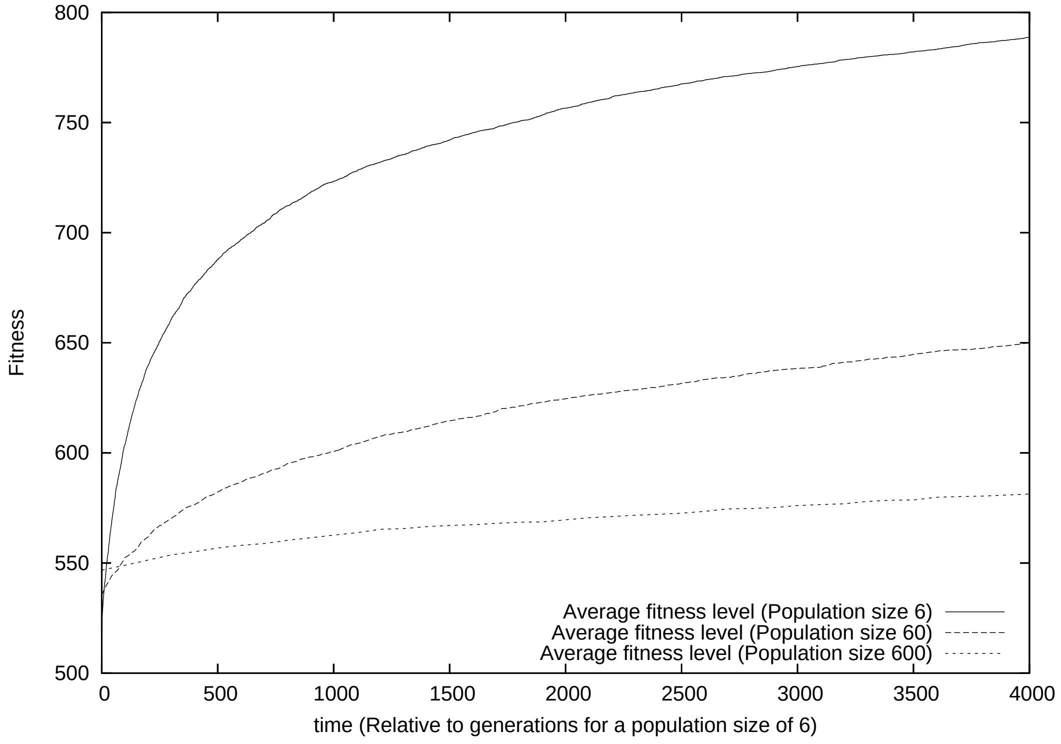

Figure 3.4: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph in Figure 3.4 shows the average best fitness of all three population sizes. They are shown against the time taken to run the GA. This graph shows that the population size of 6 does better than the higher population sizes when compared on time taken.

Population Size	6	60	600
40 Generations	557	570	576
400 Generations	676	649	—
4000 Generations	788	—	—

Table 3.2: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table 3.2 shows the average fitness levels of the three different population sizes at forty, four hundred and four thousand generations. While at 40 Generations the best results is that of the GA with a population size of 600 followed by that of the GA with a population size of 60 and the population size 6 performing the worst.

The GA with a population size of 600 has used up all the time available to it by the 40^{th} generation and as such has not been able to improve on its fitness level of 576. The GA with a population size of 60 has reached a fitness level of 649 but the GA with a population size of 6 has produced a fitness level of 676. This is interesting as the lower population size has achieved a higher fitness level than the population size of 60 in the same number of generations, which will have taken it $\frac{1}{10}$ of the time to run. This result is counter intuitive and will need to be researched more to understand why it occurs.

The GA with a population size of 60 has used up all the time available to it by the 400^{th} generation and as such has not been able to improve on its fitness level. At its 4000^{th} generation the GA with a population size of 6 has reached the highest fitness achieved.

Deceptive Trap Functions

The following graph show the results of the runs of the GA, with the varying population sizes, with the Deceptive Trap Function as a fitness function.

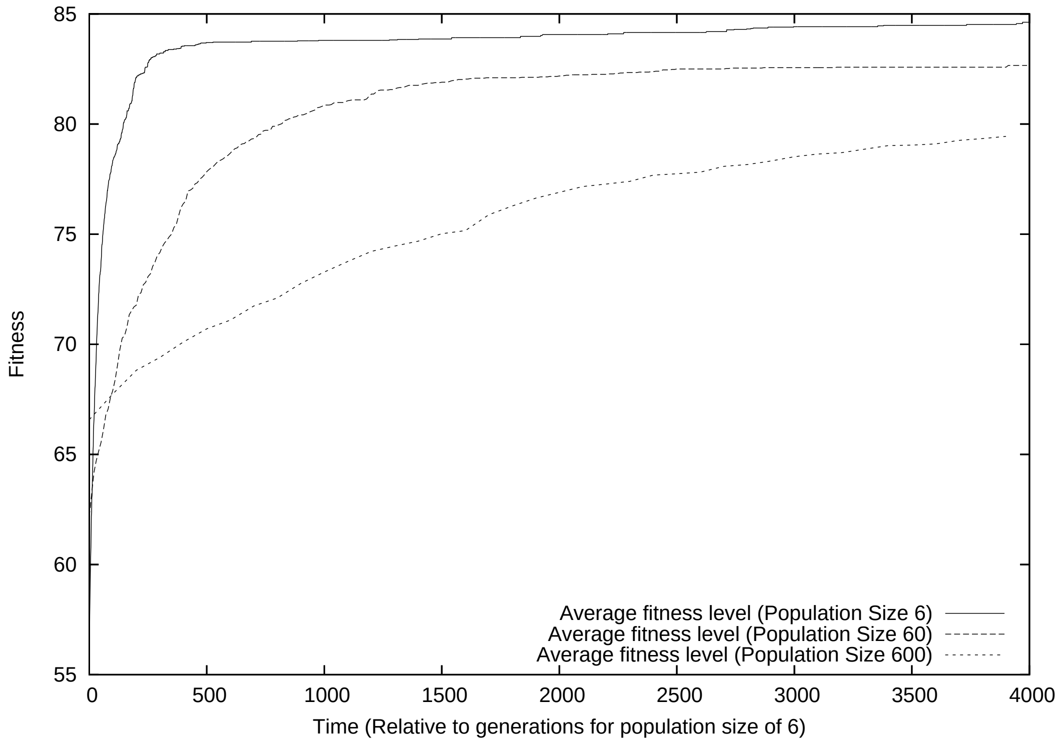

Figure 3.5: Average Fitness levels, of the best of each of 50 runs, for all three population sizes, shown against time

The graph in Figure 3.5 shows the average fitness of all three population sizes against the time taken to run the GA. Again the population size of 6 performs better than the higher population sizes in a set length of time.

Population Size	6	60	600
40 Generations	72	76	79
400 Generations	83	82	—
4000 Generations	84	—	—

Table 3.3: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table 3.3 shows that after all three population sizes had evolved for 40 generations the population size of 600 had achieved the highest fitness level.

The GA with a population size of 600 has used up all of the time available to it by 40 generations and never reaches 400 generations. So it has not been able to improve upon its fitness level of 79. Both the GAs using a population size of 6 and 60 have overtaken it at this stage. At the 400 generation stage the GA with a population size of 6 has also overtaken the GA with a population size of 60. It is important to remember that the GA with a population size of 6 has taken $\frac{1}{10}$ of the time that the GA with a population size of 60 has taken to get to this stage.

The GA with a population size of 60 has used up all the time available to it by 400 generations and never reaches 4000 generations. So it has not been able to improve upon its fitness level. The GA with a population size of 6 has been able to improve slightly again, reaching the fitness level of 84, which is the highest encountered.

The best run for the GA with a population size of 6 reached the maximum possible fitness level in the 95^{th} generation. The best run for the GA with a population size of 60 reached the maximum possible fitness level in the 45^{th} generation. While this is a difference of 50 generations, the best run of the GA with a population size of 6 reached the maximum fitness level in 3% of the time available to it while the best run of the GA with a population size of 60 reached the maximum possible fitness level in 12% of the available time.

GA Hard Problem

The following graph shows the results of the runs of the GA, with the varying population sizes, against the GA Hard Problem. The default parameters were used for the GA Hard Problem.

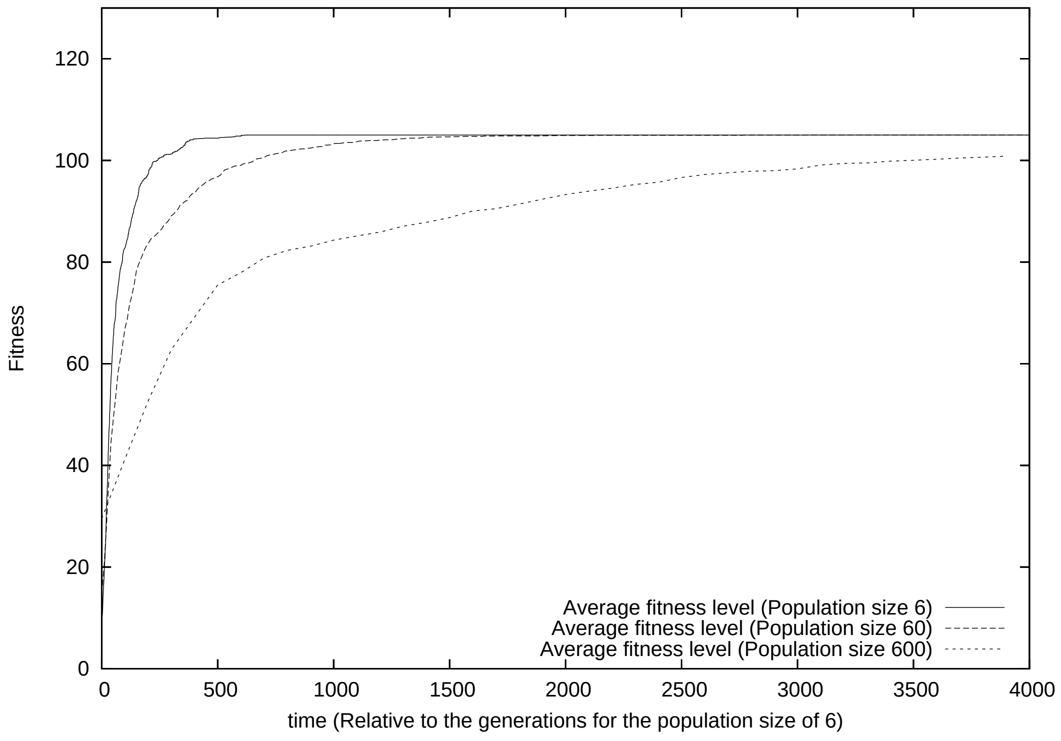

Figure 3.6: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph in Figure 3.6 shows the average fitness of all three population sizes. They are shown against the time taken to run the GA. This graph shows that the population size of 6 actually reaches the level that all the population sizes get stuck on first, while the higher population sizes take longer to get there.

Population Size	6	60	600
40 Generations	56	93	100
400 Generations	104	105	—
4000 Generations	105	—	—

Table 3.4: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table 3.4 shows the fitness levels achieved by the GA with the three different population sizes after 40, 400 and 4000 generations. After 40 generations the GAs with a population size of 60 and 600 have achieved the same level of fitness. The population size of 6 has achieved an almost equal level of fitness.

The GA with a population size of 600 has used up all the time available to it by the 40^{th} generation so never manages to improve on the fitness level of 100. Both the GAs with a population size 6 and 60 had achieved almost the same level of fitness by their 400^{th} generation.

The GA with a population size of 60 has used up all the time available to it by its 400^{th} generation so never manages to improve its fitness level above 105. The GA with a population size of 6 failed to achieve a higher level of fitness despite having enough time to run for 4000 generations.

Summary

On all three of the problems the GA using a population size of 6 performed best, on the One Max Problem and the Deceptive Trap Functions the GA with a population size of 6 scored a higher fitness value than the other two population sizes within the time limit. The GA tackling the GA Hard Problem achieved the same level of fitness with all three population sizes within the time limit but the population size of 6 reached there first.

It is interesting to note that the GA tackling the Deceptive Trap Functions managed to achieve maximum fitness on at least one of the runs when using a population size of 6 or 60 but failed when using a population size of 600. The earliest the population size of 6 achieved the maximum was on its 95^{th} generation while the earliest the population size of 60 achieved the maximum fitness level was on its 45^{th} Generation. It is important to remember that when the GA using a population size of 60 has reached its 45^{th} generation the GA using a population size of 6 has reached its 450^{th} generation.

The results show that smaller population sizes run for more generations have a better performance than a GA using a larger population size for fewer generations. This shows that when the GA is running in a fixed length of time the number of generations is more important than the population size.

3.5.2 Population Sizes and their Effects on Performance Using a Different Model of Elitism

This section shows and discusses the results of the experiments (see section 3.4.3) testing the effects of population sizes on the performance of the GA when a different model of elitism is used.

One Max Problem

The results of the experiments on the One Max Problem are shown in Figure 3.7 and Table 3.5. The graph in Figure 3.7 shows the average best fitness levels for the population sizes against time. Table 3.5 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

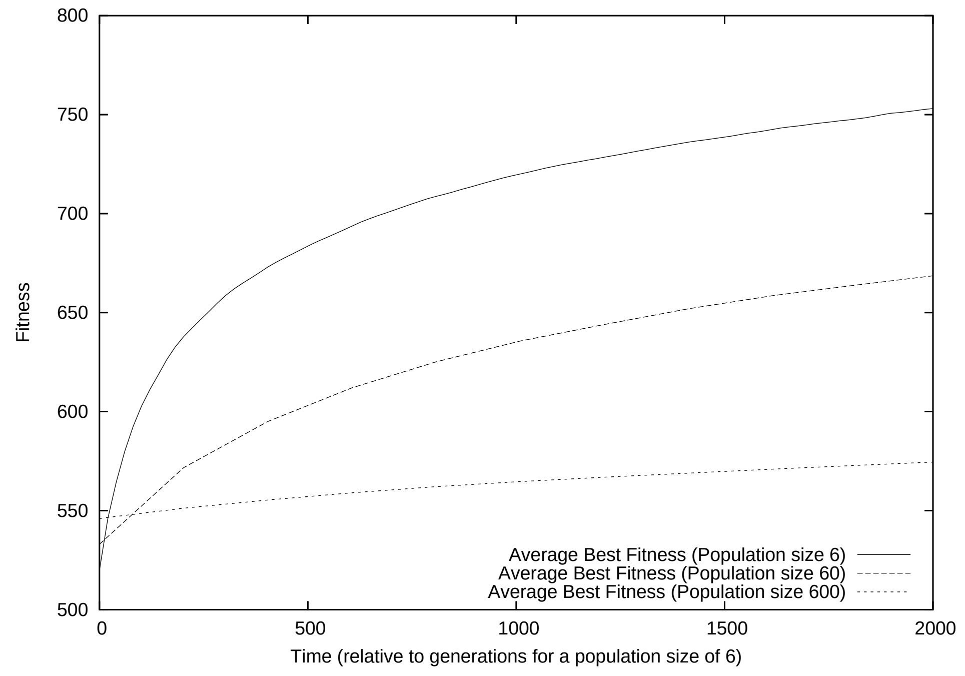

Figure 3.7: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph shown in Figure 3.7 shows that for the One Max Problem a population size 6 outperforms a population size of 60 and 600. Compared to the results shown in Figure 3.4 the population sizes of 60 and 600 have performed better, but they are still a good distance behind the results for the population size of 6.

Population Size	6	60	600
40 Generations	557	595	590
400 Generations	676	701	–

Table 3.5: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table 3.5 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. In Table 3.2 at 400 generations the population size of 6 had achieved a higher average best fitness level than that of the population size of 60 at 400 generations. This configuration of GA which carries the best $\frac{1}{6}$ forward to the next population showed a similar anomaly, where the population size of 60 has a higher average best fitness level than the population size of 600 at 40 generations. This suggests that the anomaly is related to the elitism levels the GAs are using.

Deceptive Trap Functions

The results of the experiments on the Deceptive Trap Functions are shown in Figure 3.8 and Table 3.6. Figure 3.8 shows the average best fitness levels for the population sizes against time, while Table 3.6 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

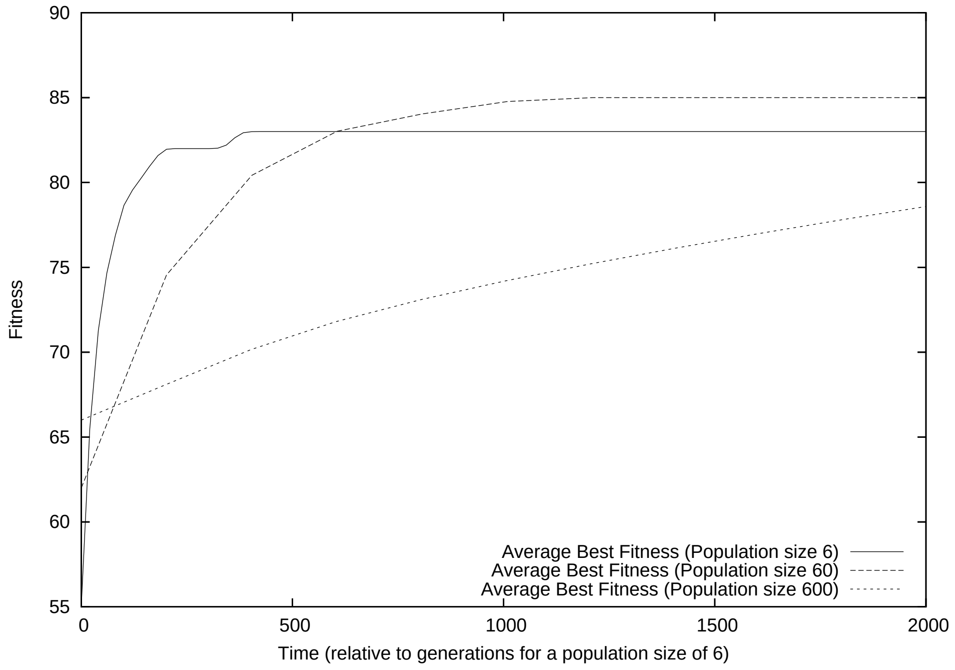

Figure 3.8: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

Figure 3.8 shows that for the GA tackling the Deceptive Trap Functions The population size of 60 outperforms the population sizes of 6 and 600. The population size of 6 performs better than the population size of 60 in the early stages, but as the time the GA has been evolving for increases the population size of 60 overtakes it.

Again compared to the results shown in Figure 3.5 the population sizes of 60 and 600 have performed better. Though the population size of 600 still does not perform as well as the population sizes of 6 and 60.

Population Size	6	60	600
40 Generations	72	81	83
400 Generations	83	85	–

Table 3.6: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table 3.6 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. Section 3.3 shows an anomaly where the population size of 6 had achieved a higher average best fitness level than the population size of 60. This anomaly is not present in the results for this configuration of GA.

GA Hard Problem

The results of the experiments on the GA Hard Problem are shown in the graph in Figure 3.9 and Table 3.7. The graph in Figure 3.9 shows the average best fitness levels for the population sizes against time, while Table 3.7 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

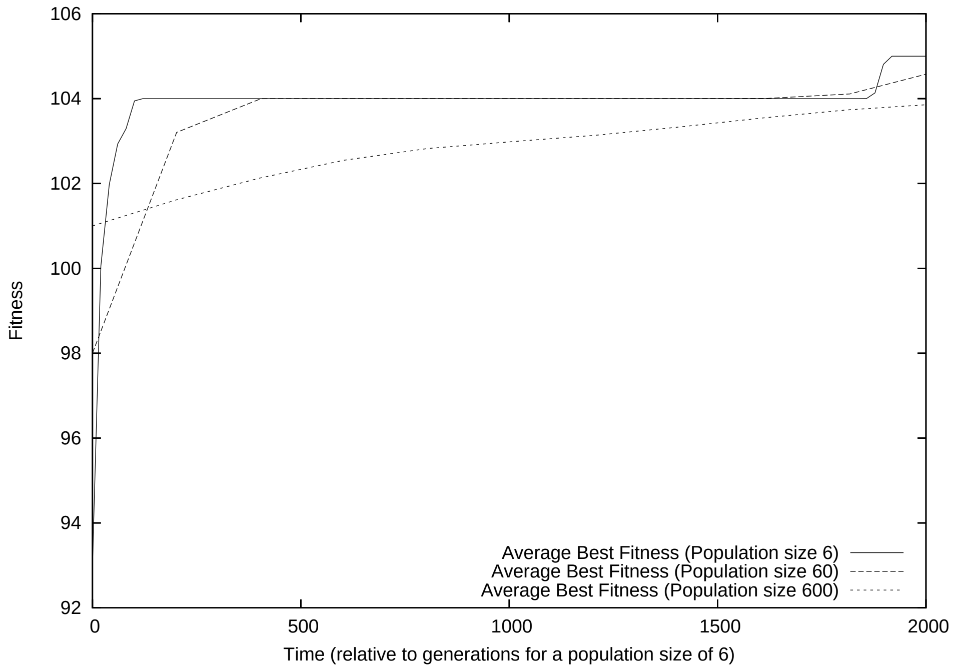

Figure 3.9: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph shown in Figure 3.9 shows that for the GA Hard Problem the population sizes of 6 and 60 are very close. The population size of 6 outperforms the population size of 60 and 600 in the time available to it.

Compared to the results shown in Figure 3.6 the results are similar with the exception of the population size of 600 which has performed better, though still not as well as the population sizes of 6 and 60.

Population Size	6	60	600
40 Generations	102	104	104
400 Generations	104	105	–

Table 3.7: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table 3.7 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. Interestingly at 40 generation the population sizes of 60 and 600 have achieved the same average best fitness level.

3.5.3 Summary

Even when a different method of elitism is used a smaller population size outperformed the larger population sizes on two of the three test problems within the given time period. The only test problem that a population size of 6 did not perform best on was the Deceptive Trap Functions where the larger population size of 60 outperformed the other population sizes.

Section 3.5.1 shows that for the Simple Elitist Genetic Algorithm a population size of 6 could outperform a population size of 60 in the same number of generations. The GA used in the experiments in this section do not show this happening which suggests that it was caused by the Simple Elitist Genetic Algorithm only carrying the best individual forward to the next generation.

3.5.4 Population Sizes and their Effects on Efficiency

This section shows and discusses the results of the experiments (see section 3.4.4) testing the effects of population sizes on the efficiency of the GA.

One Max Problem

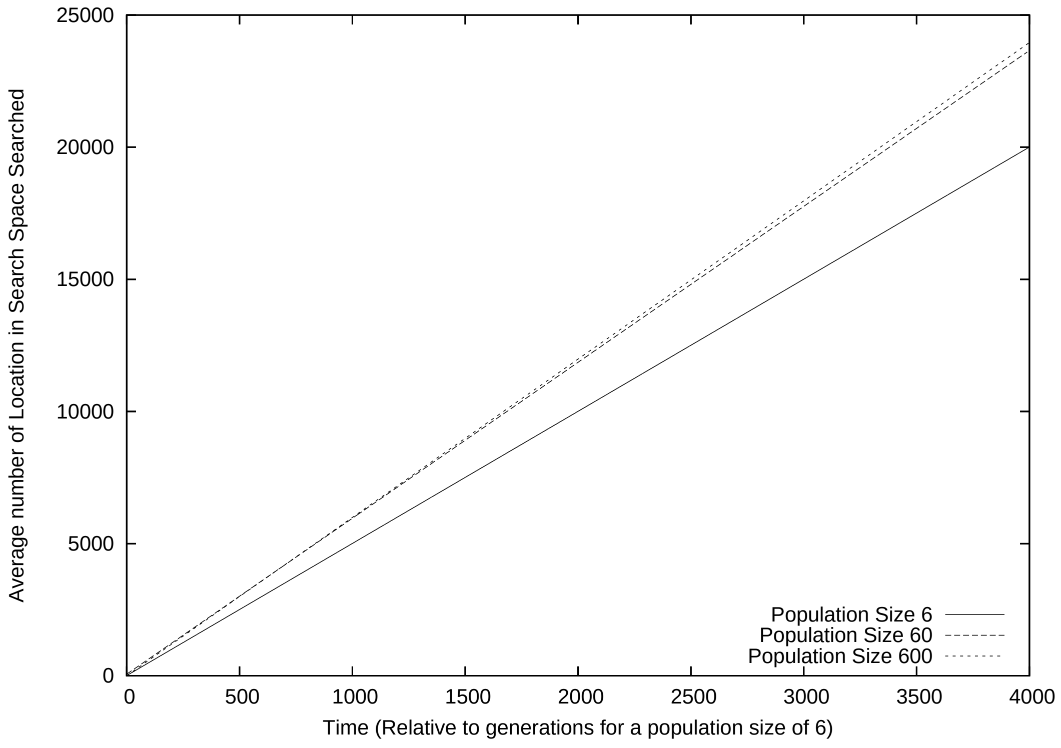

Figure 3.10: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure 3.10 shows the average number of locations in search space searched by the GAs using the three population sizes. This shows that the population size of 6 has the worst efficiency while the population size of 600 has the best. The difference between the efficiency of the population size of 60 and the efficiency of the population size of 600 is smaller than that of the difference between the population sizes of 6 and 60.

Deceptive Trap Functions

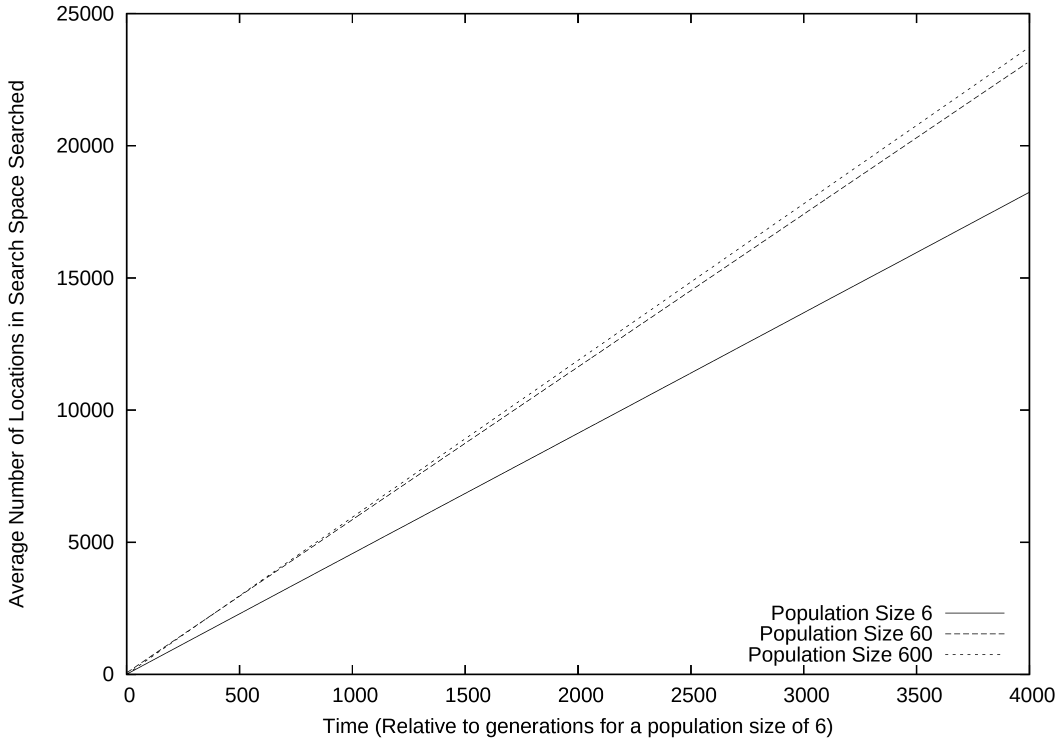

Figure 3.11: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure 3.11 shows the average number of locations in search space searched by the GAs using the three population sizes. A similar result is shown to that of Figure 3.10 with the difference in efficiency between the GA using a population size of 6 and the GA using a population size of 60 being larger than the difference between the GA using a population size of 60 and the GA using a population size of 600.

GA Hard Problem

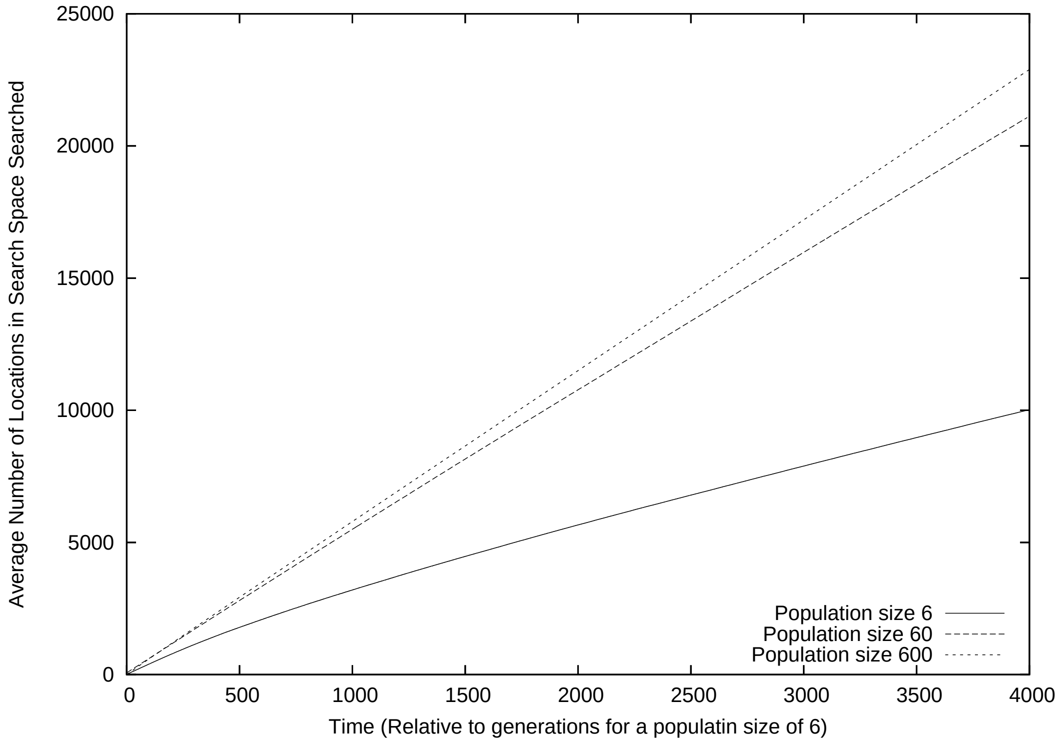

Figure 3.12: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure 3.12 shows the average number of locations in search space searched by the GAs using the three population sizes. Again the difference between the efficiency of the GA using a population size of 6 and the GA using a population size of 60 is larger than the difference in efficiency between the GA using a population size of

60 and the GA using a population size of 600.

Summary

The results show that all three GAs were at their least efficient using the population size of 6. The population size of 600 was the most efficient. This is an expected side effect of reducing the population size as doing so also reduces the variety of possible parents. Also reducing the population size increases the number of generations that the GA can run for in the same length of time. This increase in the number of generations increases the total number of individuals carried forward by elitism over the whole run.

The One Max Problem has shown that the efficiency of a GA tackling a simple problem does not suffer as much as a GA tackling a more complex problem. As the One Max Problem is easy for a GA its individuals were continually evolving into better individuals which gave a strong direction to the evolution.

As the difficulty of the problems being tackled by the GA increased the efficiency of the GA dropped. This is probably because the GA has encountered local minima which are hard for the GA to escape from, this would result in the same small area of the search space being searched.

3.5.5 Elitism Levels and their Effects on Performance

This section shows and discusses the results of the experiments (see section 3.4.5) the effects of elitism on the performance of GAs.

Deceptive Trap Functions

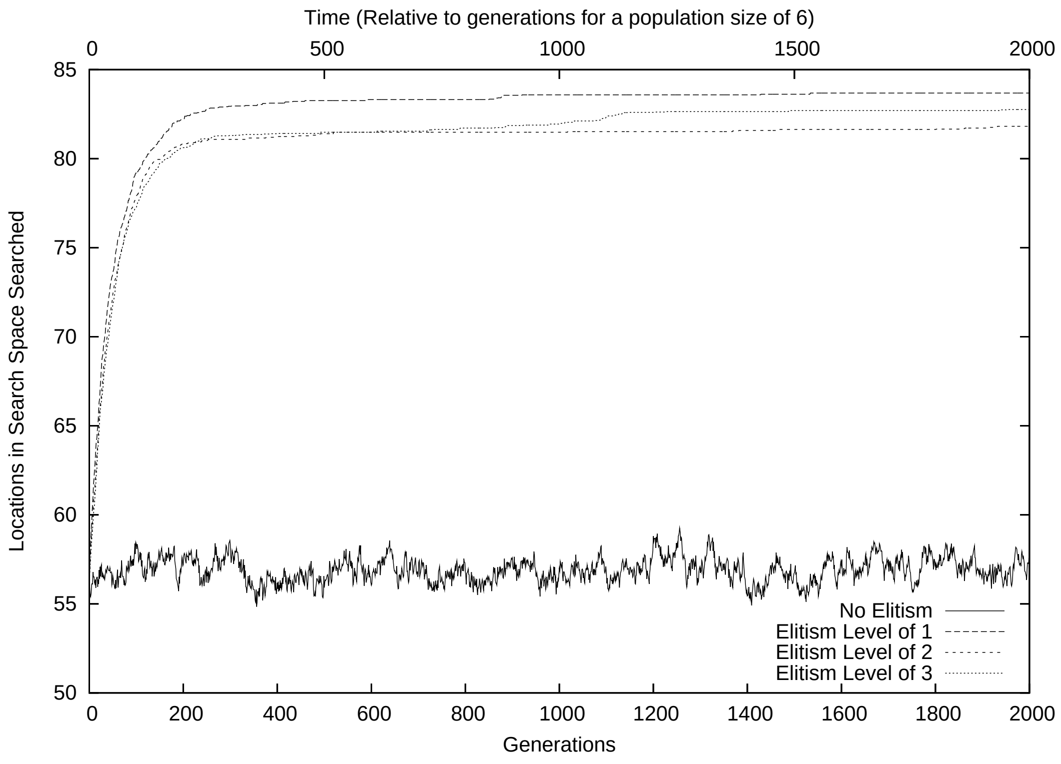

Figure 3.13: Average Fitness Levels for the Deceptive Trap Function

Figure 3.13 shows that for the Deceptive Trap Function, being tackled by the GA with a population size of 6, the best level of elitism is to carry one individual forward to the next generation. Carrying two individuals and three individuals forward produces good results but not as good as just carrying one forward, though the difference in fitness between the three levels of elitism is marginal. The GA using no elitism failed to improve.

3.6 Summary

The results show that a small population size needs a level of elitism to perform well. Without elitism the GA becomes unstable. As the gene pool is so small it is easy for the next generation to consist entirely of poor individuals. While Section 3.5.2 shows that using elitism with larger population sizes improves the performance of the GA, larger sizes of population have already been shown to operate acceptably without elitism [51].

On the simple One Max Problem the higher level of elitism performed best, on the GA Hard Problem all three levels of elitism tested got the same level of fitness in the given time, but the higher levels achieved this level marginally quicker than the lower level. On the Deceptive Trap Functions the lower level of elitism produced marginally better results. This shows that there is very little difference between a small population size using a low level of elitism and small population size using a high level of elitism when the problem is more difficult than the One Max Problem, so a standard elitism level of one will let the GA search a larger area of the search space.

In all the tests the lowest population size performed better over a fixed period of time. Even as the difficulty of the problems increased the lowest population size still performed best. This indicates that it is worth reducing population sizes when there is a limited length of time, or if it is important to reduce the length of time it takes to run the GA.

A possible reason for the population size of 6 surpassing the population size of 60 by 400 generations in the One Max Problem and the Deceptive Trap Functions is suggested by the results in section 3.5.2. As the main difference between the tests in section 3.5.2 and section 3.5.1 is how the GA implements elitism it suggests that the anomaly is caused by the way the Simple Elitist Genetic Algorithm handles the elitism.

On the simple One Max Problem, which has a smooth hill to climb, there was no problems with searching the same locations in search space multiple times. As the problem difficulty increased and local minima were introduced to the problem, the number of locations in search space that was searched multiple times increased. The problem of searching the same locations multiple times was worst in the lowest population size.

This indicates that while lower population sizes perform better when there is a fixed length of time to run the GA within, they are less efficient than the larger population sizes. If the efficiency of the GA using a low population size can be increased then they could fit even more generations into the same period of time.

The experiments testing how the level of elitism affects the performance of the GA showed that a degree of elitism is needed with a small population size as without it the GA finds it very difficult to improve. The One Max Problem, which has a smooth hill for the GA to climb, found that the best level of elitism was to carry the three best individuals forward to the next generation. The Deceptive Trap Function, which has a harder hill for the GA to climb, was tackled best by carrying only one individual forward to the next generation. The GA Hard Problem had very little difference in the performance of the GA when carrying one, two or three individuals forward to the next generation.

It is important to remember that the higher the level of elitism used then the lower the efficiency of the GA as it has more individuals carried forward that it has already seen. Unless stated otherwise the level of elitism used in the rest of this Thesis is a level of one, which provides a good compromise between the improvements to the performance achieved by elitism and the detrimental effect elitism has on the efficiency of the GA.

	Assessing Intelligent Fitness Functions	Assessing Partial Fitness Functions
Population Size	6	60
Mutation Rate	$\frac{1}{n}$	$\frac{1}{n}$
Elitism Level	1 Individual	1 Individual

Table 3.8: Base configurations used

To enable the new methods to be compared to a standard GA it is necessary to use the same base configuration for the GAs. The base configurations used in most of the experiments in this Thesis are shown in Table 3.8[2]. While these configurations have been chosen to enable the new methods to be assessed it is important to remember that these configurations have been chosen based on the performance of standard GAs and as such it may be possible to find better configurations for the new methods.

[2]The population size for assessing partial fitness functions is larger than that for assessing intelligent fitness functions because partial fitness functions require a larger population size and it would be unfair to compare them to a standard GA with a smaller sized population.

Chapter 4

Intelligent Fitness Functions

4.1 Introduction

This chapter introduces the concept of intelligent fitness functions and investigates the effect that different configurations of intelligent fitness functions have on the performance of a GA.

The longer each fitness function takes to calculate the more important it becomes for the GA to only spend time evaluating individuals it has not seen before. Intelligent Fitness Functions help a GA by reducing the number of individuals that have their fitness assessed multiple times.

4.2 Caching and GAs

One method to improve the efficiency of a GA is to use a cache. The caching GA was introduced by Kratica[83]. Kratica's caching GA checks its cache before passing an individual to the objective function for decoding. If the individual is found in the cache the objective values stored in the cache for the individual are passed to the fitness function. If the individual is not found in the cache then the objective function is used to generate the values that are passed on to the fitness function. Either way the values from the objective function for the individual are then placed into the cache. If the individual is already in the cache then the old entry is removed as this results in the cache being ordered from least recently used to the most recently used. If it is not already in the cache then it is added, if the cache is full then the oldest member in the cache is replaced.

Kratica's caching GA has had varied results depending on the problems being tackled [83]. The results range from a speed up of the GA of 0.23% as the worst reported result, to a speed up of 98.3% for the best reported result (Kratica did not state how a speed up of 98.3% was achieved by the caching GA in his report [83]). The average speed up of a GA reported by Kratica [83] was 23.3%.

To use this design of caching GA an existing system would need to be modified to add the caching process around the objective function, which may help explain why it is not widely used. The caching GA uses a least recent used replacement strategy for the cache which is the most commonly used strategy for caching [143] but no evidence has been provided that it is the best for use with a GA.

The caching GA has been used to tackle the following problems.

- The Simple Plant Location Problem [84, 82]
- The Index Selection Problem [85, 24]
- The Edge-Biconnectivity Augmentation Problem [92]

4.3 Concept

A standard GA fitness function takes a genome, decodes it into a phenotype and then uses the phenotype to calculate the fitness of the genome. Every genome goes through the same process. In this Thesis intelligent fitness functions consist of an active control part and either a long term memory, a short term memory or both. The control is then able to check whether an identical individual has been encountered before. The control then decides how to proceed based on the information it has about an individual.

Intelligent Fitness Functions differ to the Caching GA in three areas. Firstly the Caching GA only caches the result of the objective function while intelligent fitness functions remember the result of the fitness function While it is a simple function to calculate the fitness from an individuals objective value, the objective value is not used anywhere else in the system except to calculate the individuals fitness. A further saving can then be achieved by remembering the fitness value of an individual instead of the objective value. Secondly the caching in the Caching GA is a property of the GA itself while the memory of the intelligent fitness functions is stored within the fitness function. This enables a GA to benefit from the use of an intelligent fitness function without the need to modify the GA's engine. The third area that intelligent fitness functions differ to the caching GA is that the intelligent fitness functions can use a short term memory as well as a long term memory. This enables the intelligent fitness function to treat an individual that is a duplicate of another individual encountered in the same generation differently to an individual that is a duplicate of an individual encountered in a previous generation.

4.3.1 Memory Types

There are two types of memory an intelligent fitness function can contain.

Short Term Memory enables the intelligent fitness function to remember the individuals it has seen in that generation. The goal of the short term memory is to increase the diversity of the GA's population.

Long Term Memory enables the intelligent fitness function to remember individuals it has seen in previous generations. The goal of the long term memory is to reduce the number of repeated fitness tests.

Short Term Memory

If a genome is identical to another one in the current population, i.e. the fitness function finds the individual in its short term memory, then it gives that individual a fitness of 0 to try to increase the variety of the next generation by reducing the chances that identical individuals will be chosen as parents for a new individual. This is intended to stop an individual from taking over the population. The diagram in Figure 4.1 shows an intelligent fitness function which contains a short term memory as well as a long term memory.

Long Term Memory

If an individual has already been tested in a previous generation and the intelligent fitness function finds it in its long term memory, then it can simply return the fitness level the individual scored last time it was evaluated. The diagram in 4.1 shows an intelligent fitness function which contains a long term memory as well as a short term memory.

If a GA is run for a large number of generations it may not be efficient to remember every individual encountered, as the intelligent fitness function will have to search the long term memory every time it is asked to assess the fitness of an individual. To avoid the inefficiency that could arise from storing every individual the long term memory should have a limit to the number of individuals it can store (this limit is discussed later in this Chapter, see section 4.3.3).

Combined Long Term and Short Term Memory

The short term memory is designed to increase the diversity of the GA's population, while the long term memory is designed to avoid re-evaluating identical individuals. Due to the difference in the designed effect of the long term and short term memories it is possible to combine them together in the same intelligent fitness function. If they are combined together the short term memory would normally take priority over searching the long term memory. A diagram of an intelligent fitness function with both a long term and a short term memory is shown in Figure 4.1.

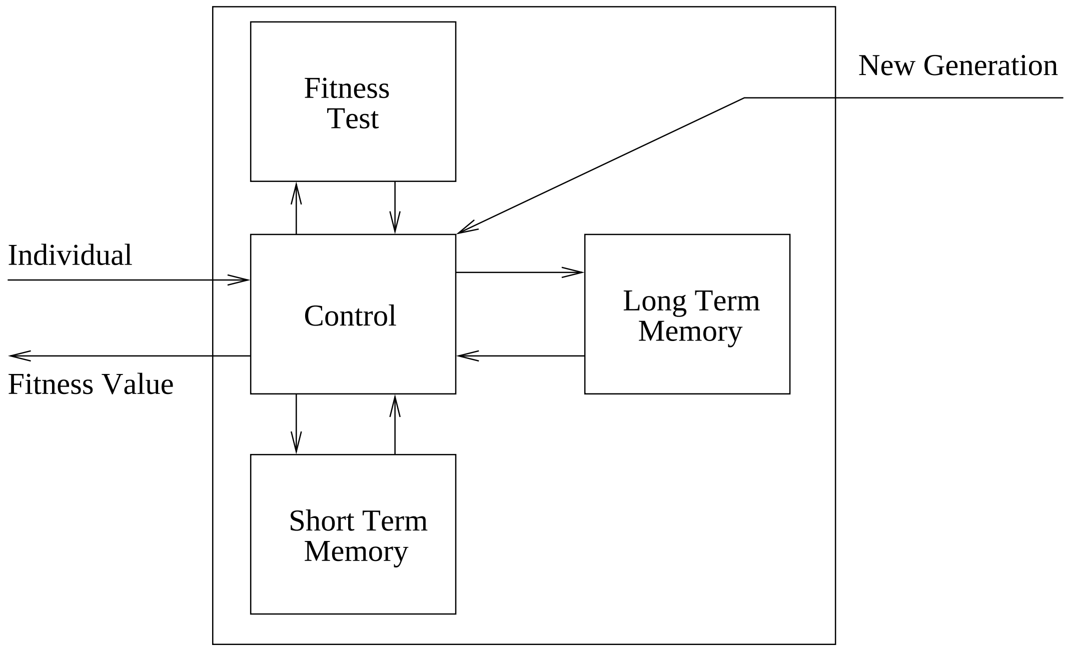

Figure 4.1: An Intelligent Fitness Function with Both a Long Term Memory and a Short Term Memory

4.3.2 Memory Search and Replace Times

An intelligent fitness function will need to spend time searching through its available memories to check if it has information about the individual it has been asked to assess. Also each time an intelligent fitness function is called it needs to update its memory which also takes time. Equation 4.1 shows how to calculate the average time taken to calculate the fitness of an individual using an intelligent fitness function. If the value of AF is less than the time taken to calculate the fitness of an individual using a standard fitness function then it is beneficial to use an intelligent fitness function.

$$AF = \left(\left(\frac{M}{2} + U\right) H\right) + (((M + F) \, . \, (1 - H)) + M + W + U) \tag{4.1}$$

AF Average length of time for the intelligent fitness function to calculate and individual's fitness

M Length of time taken to search the memory

H Expected chance of locating an individual in the memory

F Length of time taken to calculate the fitness of an individual

W Length of time taken to write an individual and its fitness value into the memory

U Length of time taken to update the status of an individual in the cache

From Equation 4.1 we can see that as long as $\frac{M}{2}$ is less than F, as the length of time taken to search the memory increases the savings to be made decrease. Yet as the chance of an individual being found in the memory increases so too do the savings. Also as the length of time it takes to calculate an individual's fitness increases so too does the available savings from using an intelligent fitness function.

The parameter H in equation will be dependent on the problem being tackled and the configuration of the GA. For a Simple Elitist Genetic Algorithm with a population size of 6 a value of $\frac{1}{3}$ can be used as an approximation (Taken from the results of the experiments in the rest of this chapter). To obtain a more accurate value then the GA being used and the effects the problem being tackled have on it would need to be investigated.

The length of time taken to update the status of an individual in the cache is not always needed. If the storage strategy (see Section 4.3.4) is based on the individual's fitness then that will not need to be changed each time it is encountered. Storage strategies that track the number of times an individual has been encountered or the time it was last encountered will need to update their cache.

For the test problems used in this chapter it takes longer to search the intelligent fitness functions memories than it does to calculate an individual's fitness. As the experiments are exploring the effects that the intelligent fitness functions have on GAs these simple test problems are preferred.

4.3.3 Memory Sizes

The size of memory needed for the short term memory is shown in Equation 4.2. The memory needs to be this size as it has to be capable of storing every unique individual seen in that generation, though it doesn't need to store the last individual in the generation as it will be cleared for the start of the next generation. In most cases it is easier to have a short term memory size equal to the size of the population as then there is no need to distinguish between the last individual in a generation and all the other individuals. The only time lost this way is the time taken to write the last individual to the memory as the memory is cleared straight afterwards.

$$SM = P - 1 \tag{4.2}$$

SM size of the short term memory needed.

P size of the GA's population.

The size of memory for the long term is not as easy to decide upon as the size of the short term memory. Equation 4.1 shows that as the length of time taken to search the memory increases the savings to be made by the intelligent fitness function decrease. The experiments in the rest of this chapter will test a range of long term memory sizes as well as the type of long term storage strategy used.

4.3.4 Long Term Memory Storage Strategies

As the long term memory is limited a strategy is needed to decide which individuals to replace with the new individuals being stored. There are four obvious criteria to base the decision on.

Highest Fitness Based is where the long term memory is used to store the individuals encountered so far with the highest fitness. New individuals with high fitness will replace the least fit individual in the long term memory. As the fitness of an individual increases so too does the chance that it will be selected as a parent. This would suggest that the higher fitness individuals have more chance of appearing in the next generation.

Lowest Fitness Based is the opposite of the Highest Fitness Based strategy. The least fit individuals encountered are remembered. This at first appears to be of little use as a strategy but if a GA gets stuck for a time at a local minimum then it will encounter a lot of individuals with a lower fitness before it manages to find a better area to search.

Time Based is a strategy that stores individuals based on how long ago it has been since they were last encountered. New individuals to be stored in the long term memory replace the oldest individuals. As each generation is based on the previous generation an individual is more likely to be identical to one in the previous generation than one in the first generation.

Frequency Based The long term memory can be used to store individuals that have been encountered many times. The new individuals will replace the least seen individual in the long term memory. If an individual keeps appearing it is likely to keep doing so.

4.4 Experiments on the Effects of Intelligent Fitness Functions

Two problems were tackled with an intelligent fitness function system that had the options of a short term memory, long term memory or both. All 4 storage strategies were assessed. The first GA had the Deceptive Trap Function (See 3.3.2) as its fitness test while the second GA had the GA Hard Problem (See 3.3.3) as its fitness test. As the One Max Problem (See 3.3.1) did not re-evaluate many individuals, the intelligent fitness functions would not be able to make an improvement.

The GA used a population size of 6, a mutation rate of $\frac{1}{n}$ [51], single point crossover and used elitism that carried one individual forward to the next generation. These settings have been shown in Chapter 3 to be good for a Simple Elitist Genetic Algorithm. While this configuration may not be the best for the GAs being tested the use of a consistent configuration enables the results of the GAs to be compared and the level of improvement provided by using intelligent fitness functions can be measured.

4.4.1 Experiment Details

Each GA was run using the same set of 50 seeds for the pseudo random number generator and the results were then averaged for the graphs of the results. For each seed the GA was run with the following settings.

- Standard fitness function (No memory)
- Short Term Memory only
- Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the highest fitness based storage strategy
- Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the lowest fitness based storage strategy
- Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the time based storage strategy
- Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the frequency based storage strategy
- Short Term Memory and Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the highest fitness based storage strategy
- Short Term Memory and Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the lowest fitness based storage strategy
- Short Term Memory and Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the time based storage strategy

- Short Term Memory and Long Term Memory with sizes from 6 to 60, in multiples of 6 individuals, using the frequency based storage strategy

4.4.2 Results

In this Section each graph shows how a specific long term memory storage strategy performed on the problem being tackled. Each graph shows four lines.

Long Term Memory Only shows the savings achieved using only the long term memory with the storage strategy being examined.

Short Term Memory Only shows the saving achieved by using only the short term memory

Sum of Short Term Memory Only and Long Term Memory Only shows the expected result if the use of a short term memory and the long term memory storage strategy being examined is accumulative.

Short Term Memory and Long Term Memory shows the actual savings achieved by using the short term memory and long term memory together for the storage strategy being examined.

Deceptive Trap Functions

The graphs in Figures 4.2 to 4.5 shows the improvement of the efficiency of the GA using the four different storage strategies for the long term memory.

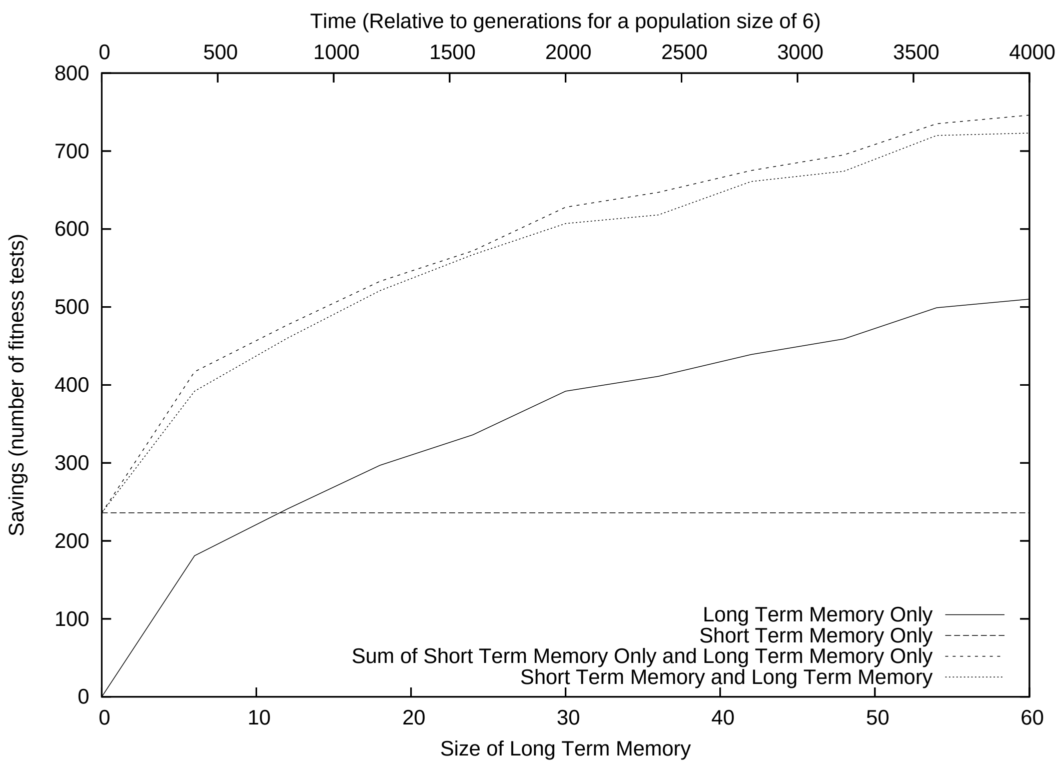

Figure 4.2: Comparison of Highest Fitness Based Storage Strategy

The graph in Figure 4.2 shows that the use of a highest fitness based storage strategy does improve the efficiency of the GA. The use of a short term memory with the long term memory produces results similar to that of the results for the long term memory added to the results of the short term memory. The graph also shows that as the size of the long term memory increases so does the impact the long term memory has on the efficiency of the GA. It is possible that increasing the long term memory beyond 60 will achieve further improvements although the gains appear to be diminishing.

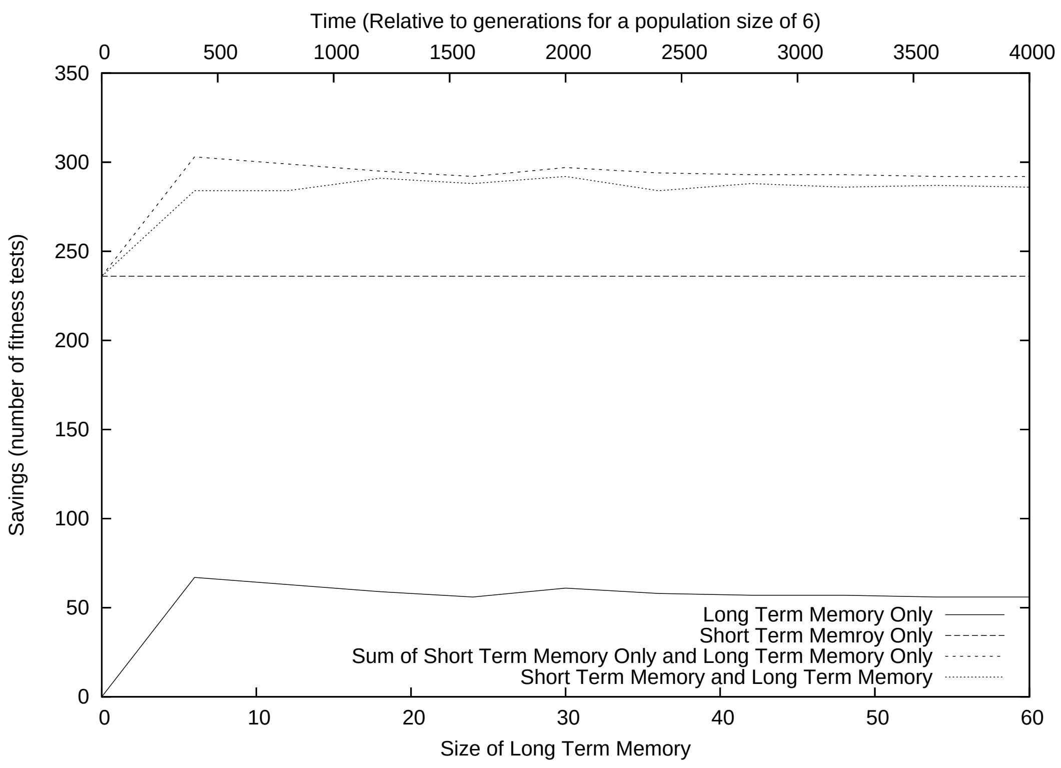

Figure 4.3: Comparison of Lowest Fitness Based Storage Strategy

Figure 4.3 shows that the use of a lowest fitness based storage strategy only improves the efficiency of the GA by a small amount. Again the use of both long term memory and short term memory gives results similar to that of adding the short term memory results to the long term memory results. The graph also shows that beyond a limited point increasing the size of the long term memory does not increase the effect the long term memory has on the efficiency of the GA.

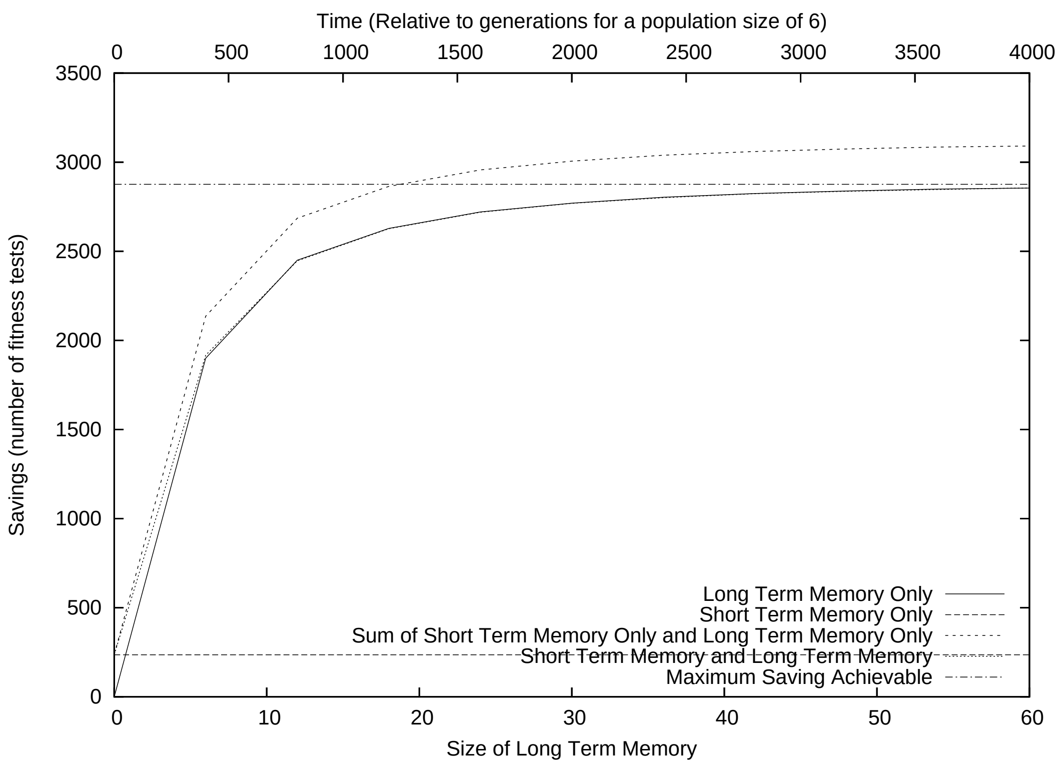

Figure 4.4: Comparison of Time Based Storage Strategy

The graph in Figure 4.4 shows that the time based storage strategy has the greatest increase in the efficiency of the GA out of the four strategies. The effect of using both long term with the time based storage strategy and a short term memory is effectively the same as just using the long term memory. Though after a long term memory size of 18 the suggested savings of the sum of the long term memory and the short term memory are impossible for the GA to achieve as they have passed the maximum savings available. The increase in efficiency increases less as the size of the long term memory increases. Compared to other approaches these improvements are considerably greater. This will be discussed later.

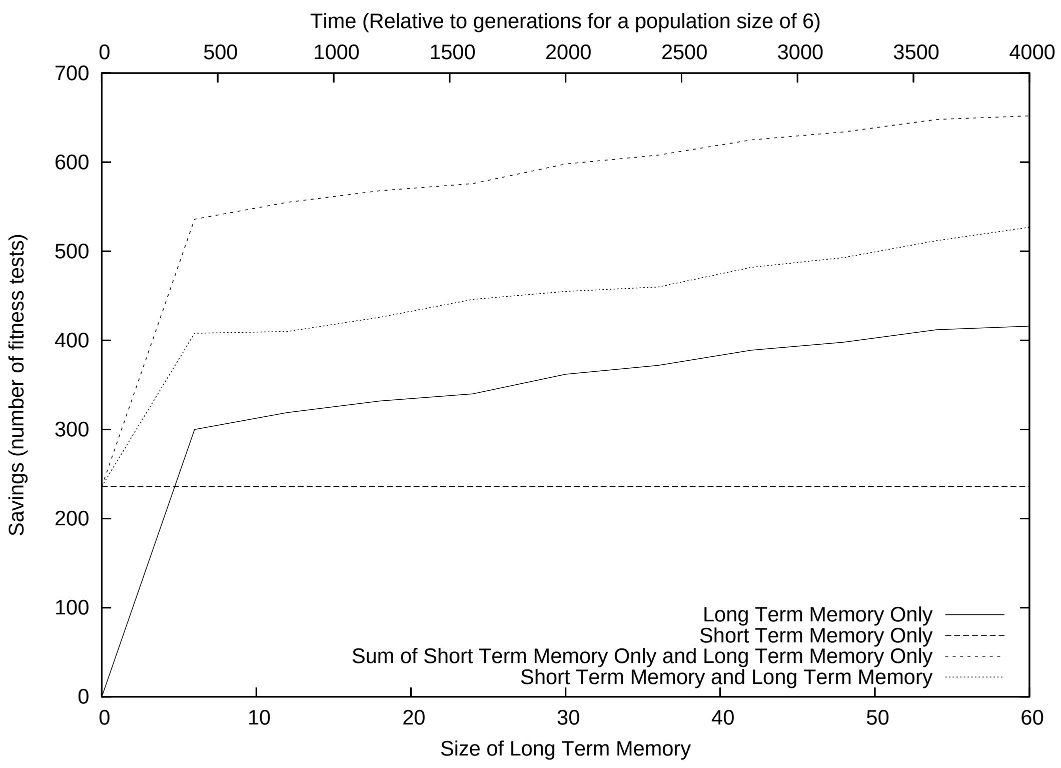

Figure 4.5: Comparison of Frequency Based Storage Strategy

Figure 4.5 shows the results of using the frequency based storage strategy. The long term memory improves the efficiency of the GA. The use of the short term memory as well as the long term memory does improve the efficiency but only by about half of that which is predicted by adding the results of the long term memory to the results of the short term memory. The efficiency of the GA increases as the size of the long term memory is increased.

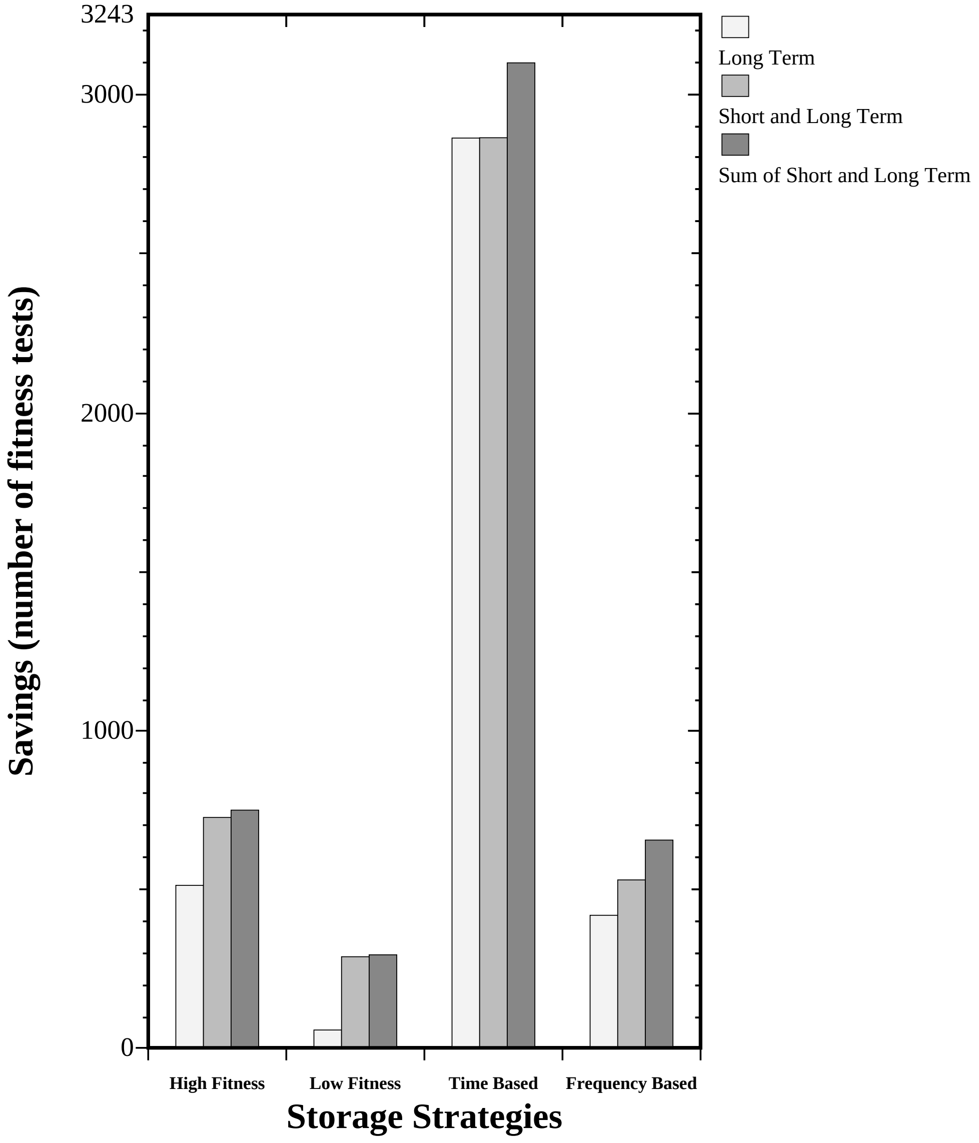

Figure 4.6: Summary of long term memory storage strategies (using a long term memory size of 60) for the Deceptive Trap Functions.

The graph shown in Figure 4.6 summarises the results of the experiments. The results of the time based storage strategy stand out as being superior by far to the other three storage strategies. It also shows that for the high fitness based, low fitness based and frequency based storage strategies the short term memory helped improve the efficiency of the GA. The time based storage strategy using a long term and short term memory produced almost identical results as the time based storage strategy just using a long term memory.

GA Hard Problem

The graphs in Figures 4.7 to 4.10 shows the improvements in the efficiency of the GA, tackling the GA hard problem, using the four different storage strategies for the long term memory.

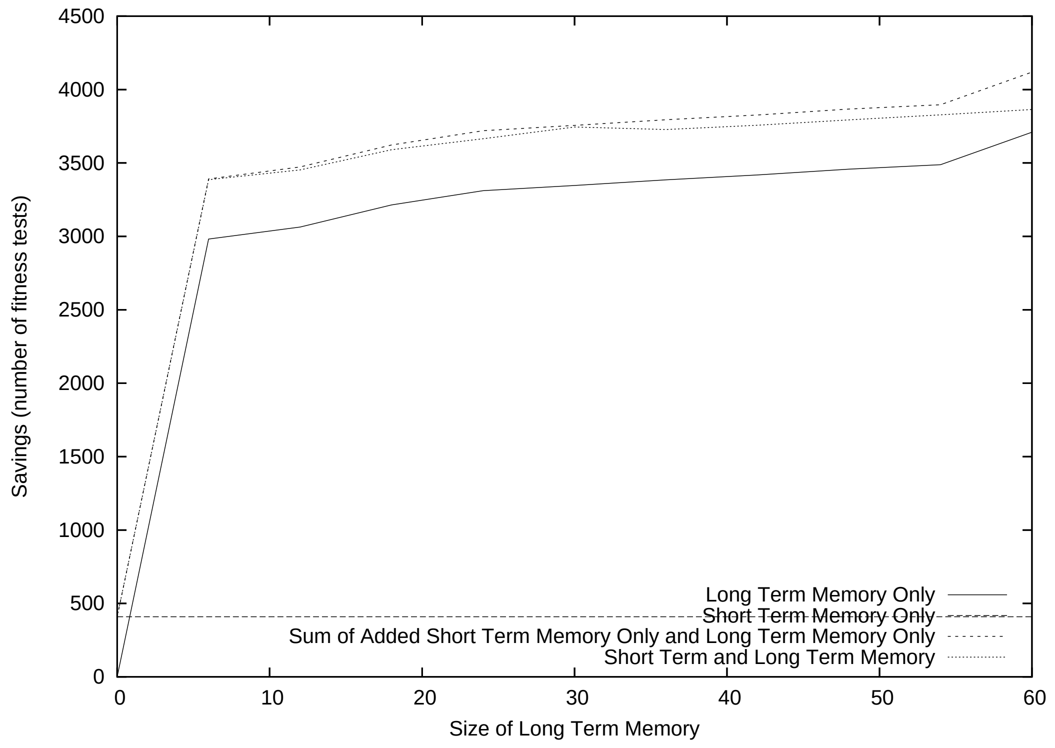

Figure 4.7: Comparison of High Fitness Based Storage Strategy

The graph in Figure 4.7 shows that the High Fitness based storage strategy does improve the efficiency of the GA. The use of long term memory combined with short term memory results in very similar savings to those suggested by adding the short term memory results to the long term memory results. The graph also shows that as the size of the long term memory is increased so too does the efficiency of the GA. The biggest improvement occurs at the initial point of applying the intelligent fitness function with savings of almost 3000 fitness tests.

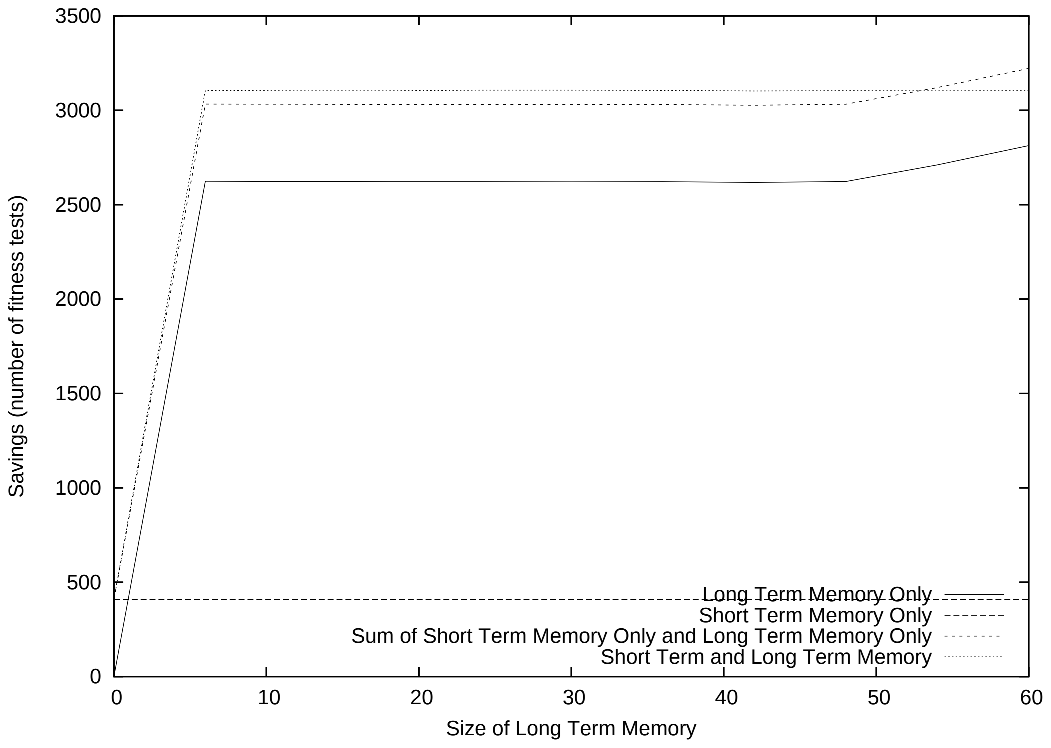

Figure 4.8: Comparison of Low Fitness Based Storage Strategy

Figure 4.8 shows that on the GA hard problem, long term memory using a low fitness based storage strategy performed better than the short term memory. The results of combining the short term memory and the long term memory using a low fitness based storage strategy is not as high as would be expected if the effects were cumulative. There is no increase in the effects of the long term memory as the size of the memory is increased until it reaches 60 where there is a noticeable increase in the efficiency.

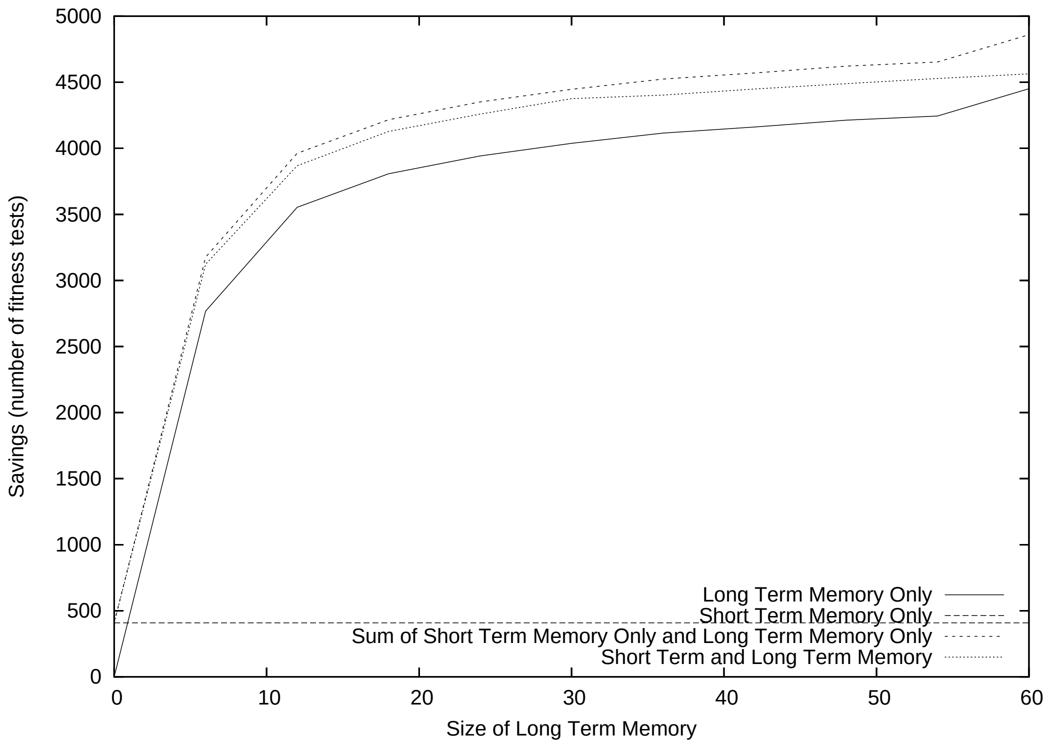

Figure 4.9: Comparison of Time Based Storage Strategy

The graph in Figure 4.9 shows that the time based storage strategy produces a large increase in the efficiency of the GA. As the size of the long term memory increases the less effect it has on the improvement of the GA. There is very little difference between the results with a long term memory size of 30 individuals and that of 54 individuals. Again there is a sudden increase in the savings achieved when a long term memory size of 60 is used.

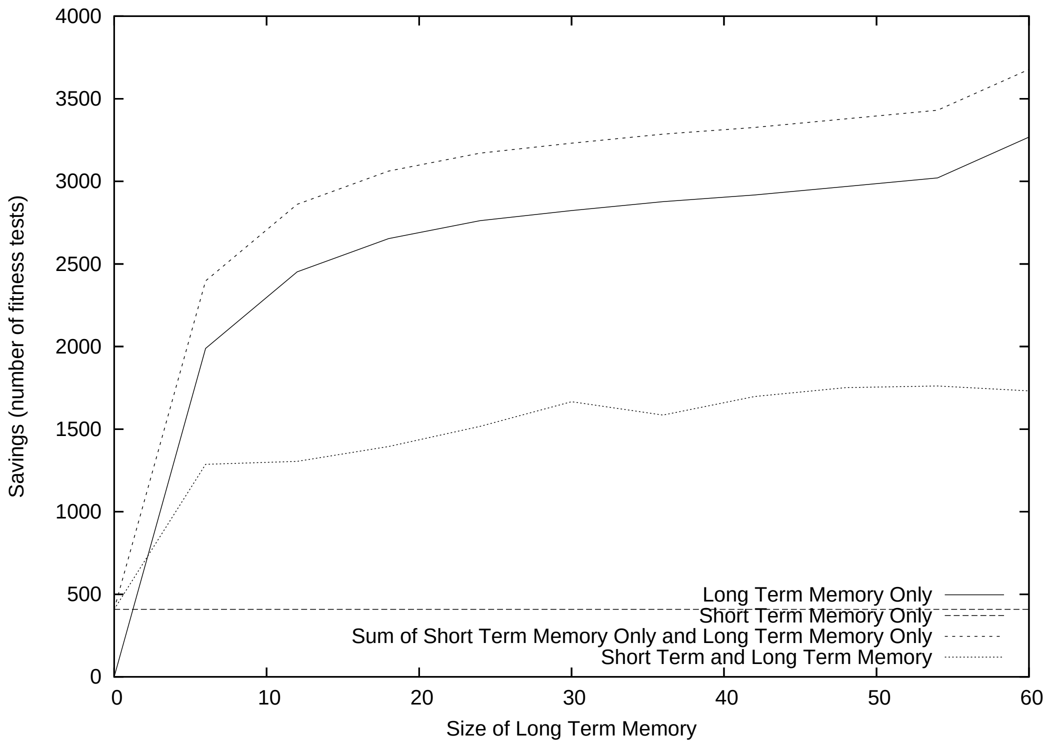

Figure 4.10: Comparison of Frequency Based Storage Strategy

Figure 4.10 shows that the frequency based storage strategy's effectiveness increases as the size of the long term memory is increased. The addition of the short term memory halves the savings achieved by the frequency based long term memory. This is discussed later in at the end of this section.

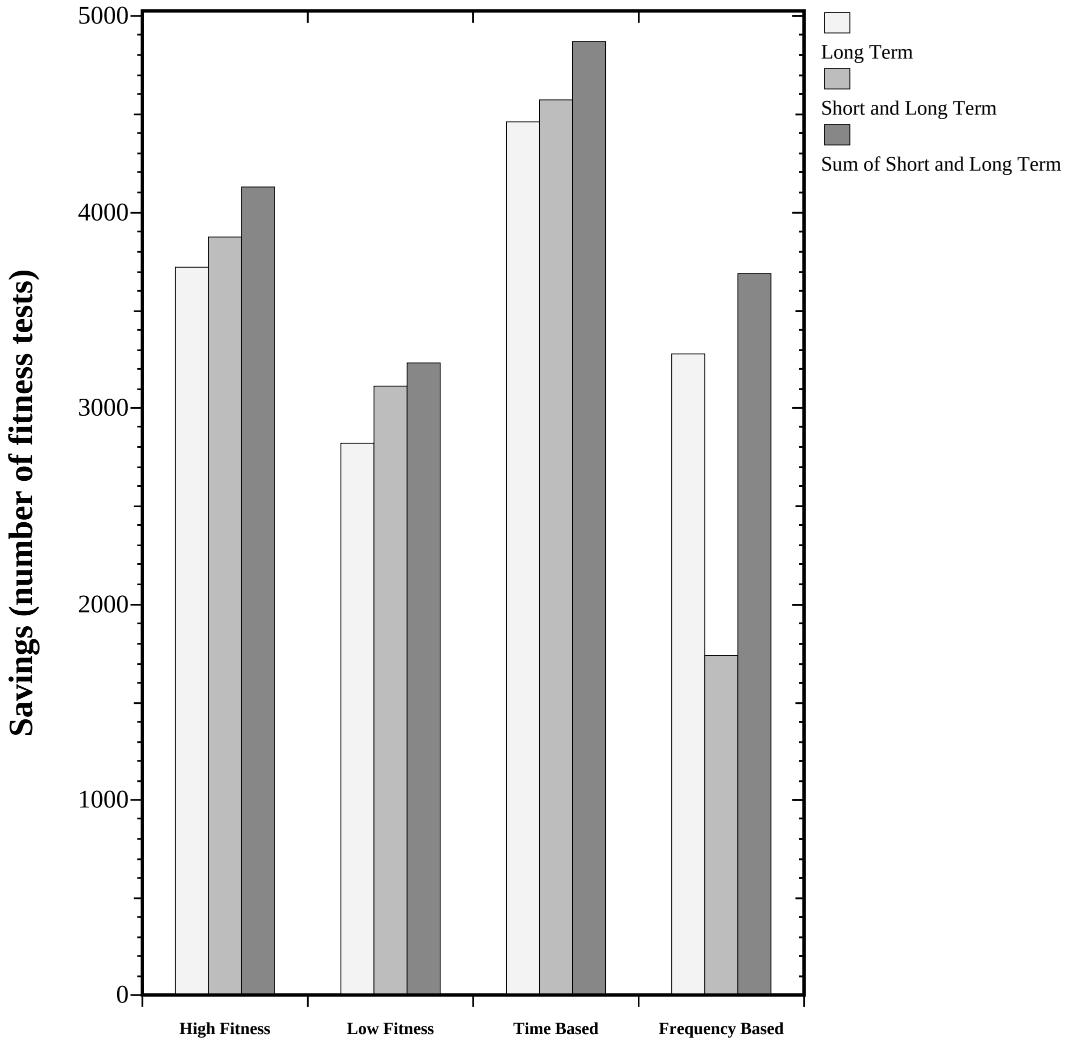

Figure 4.11: Summary of long term memory storage strategies (using a long term memory size of 60) for the GA Hard Problem

The graph in Figure 4.11 shows a summary of the results obtained for the GA tackling the GA Hard Problem. The worst savings were achieved by the GA using the frequency based long term storage strategy combined with a short term memory. This is understandable as the short term memory is trying to lower the number of duplicates carried forwards to the next population. This interferes with the frequency based storage strategy. The frequency based storage strategy gets a distorted view of the number of times an individual has already been encountered because it only gets to see the individuals not dealt with by the short term memory.

With no short term memory the worst performance was from the low fitness based storage strategy. The frequency based storage strategy performs better than the low fitness based storage strategy when it doesn't have the short term memory.

The time based storage strategy again provided the best increase in the efficiency of the GA, though the high fitness based storage strategy was a lot closer.

All four strategies have achieved savings above 2500 whereas with the GA tackling the Deceptive Trap Functions only one strategy achieved savings above 1000. The increased savings by all of the strategies is probably due to the problem being harder and more of the individuals encountered by the GA have been seen by the GA before.

4.4.3 Experiments on a Larger Population details

The same two problems were tackled by an intelligent fitness function system, using a population size of 60, to assess the usefulness of using intelligent fitness functions on larger population sizes. Having established in Section 4.4 that a long term memory with a time based storage strategy was most efficient this was the strategy used. The long term memory sizes of 30, 60, 90, 120, 150, 180, 210, 240, 270 and 300 were tested. Each memory size was run 50 times and the results averaged for comparison. Each set of runs used the same 50 seeds for the pseudo random number generator.

4.4.4 Results of Larger Population Experiments

This section shows the results of the experiment on using a population size of 60.

Deceptive Trap Problem

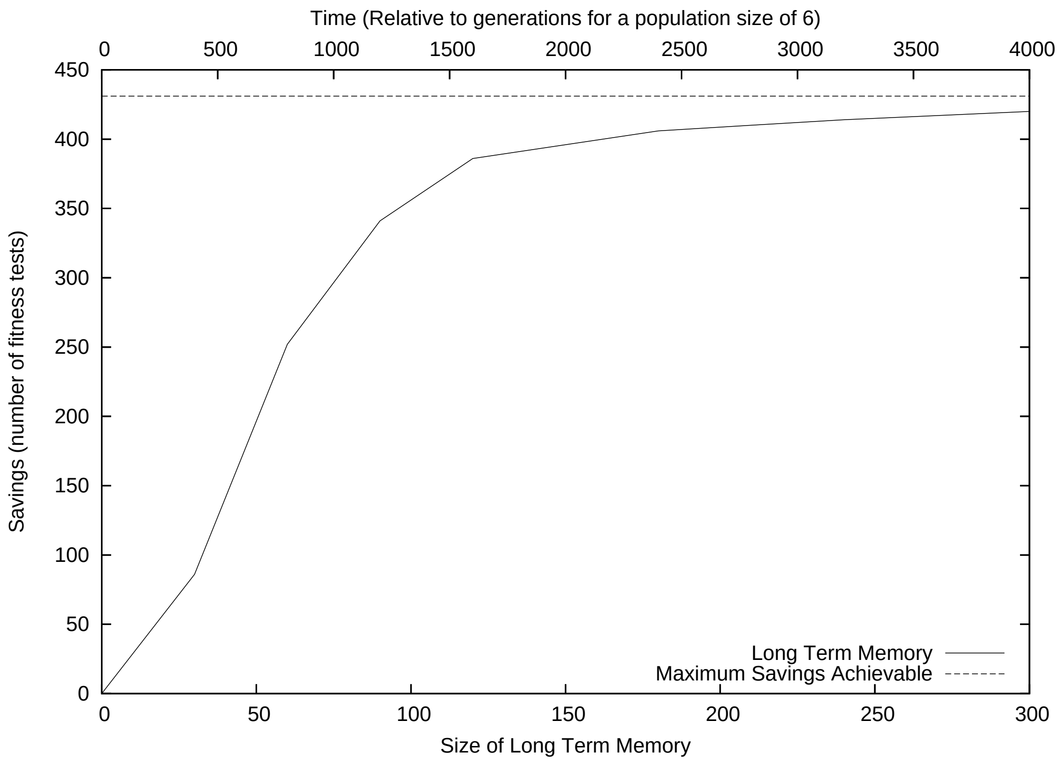

Figure 4.12: Savings made on a GA tackling the deceptive trap problem, by using an intelligent fitness function with a population size of 60

The graph in Figure 4.12 shows that, on the deceptive trap problem, the savings made with a population size of 60 are less than the savings made by the same intelligent fitness function on a population size of 6. This is due to the larger population size, which as shown in Chapter 3 encounters fewer duplicates.

GA Hard Problem

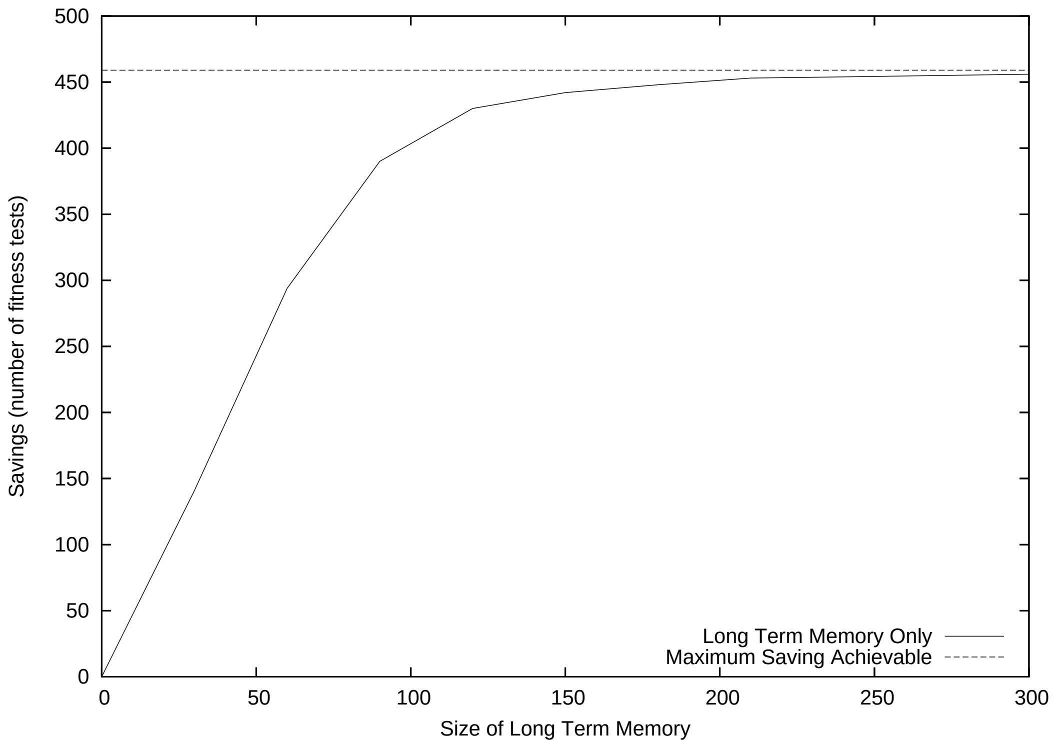

Figure 4.13: Savings made on a GA tackling the GA hard problem, by using an intelligent fitness function with a population size of 60

Figure 4.13 shows, on the GA hard problem, the savings made with a population size of 60 are again fewer than the savings made by the use of an intelligent fitness function on a population size of 6.

4.5 Summary and Conclusion

The use of an intelligent fitness function can improve the performance of a GA. Both the long term and short term memories helped improve the efficiency of the GA. In both problems the best long term storage strategy was the time based storage strategy, while the worst for the Deceptive Trap Problem was the lowest fitness based storage strategy and for the GA Hard Problem the high fitness based storage strategy was worst.

Intelligent fitness functions are not suitable for classes of problems where the results of an individual can change over time. This is because the individuals fitness will need to be freshly calculated every time it is seen. Also intelligent fitness functions are not suitable for problems where it is quicker to calculate the fitness of an individual than it is to handle the overheads of maintaining and searching the memory of the intelligent fitness function.

There was a large difference between the four storage strategies used for the long term memories. The high fitness based storage strategy steadily improved the efficiency as the size of the long term memory was increased. It benefited greatly from the addition of short term memory as well.

The use of a lowest fitness based storage strategy gave poor results, on the Deceptive Trap Functions, compared to the other strategies. Increasing the size of the long term memory only gave benefits up to a point, after that point the increase in long term memory size did not have an effect. The inclusion of a short term memory benefited the lowest fitness based storage strategy.

The time based storage strategy was the best out of the four. It also performs well with a small size of long term memory. Once the long term memory reached 30 individuals the benefits from an increase were very small. The addition of a short term memory had very little effect on the results for the GA tackling the Deceptive Trap Functions but it did have a noticeable effect on the GA tackling the GA Hard Problem. It is important to note though that the time based storage strategy achieved almost maximal savings possible for the GA tackling the Deceptive Trap Functions which helps explain why the short term memory may not have had as much effect on the GA. It also achieved 70% of the available savings for the GA tackling the GA Hard Problem. The benefits provided by the short term memory, when combined with the time based storage strategy, are problem dependant, providing very little benefit on the Deceptive Trap Problem while showing more benefits for the GA Hard Problem.

The frequency based storage strategy did not perform as well as the time based storage strategy or the highest fitness based storage strategy on the Deceptive Trap Functions but it did outperform the lowest based storage strategy. On the GA Hard Problem the frequency based storage strategy was again better than the low fitness based storage strategy. As the size of the long term memory was increased so too did the improvement in the efficiency of the GA. The short term memory hindered the frequency based storage strategy tackling the GA Hard Problem.

The use of a long term memory using a time based storage strategy greatly improves the efficiency of a GA. As the performance of the short term memory is problem dependant each problem will have to be investigated before it is known if short term memory will be useful. Even with a population size of 60 a GA can benefit from the use of an intelligent fitness function using a time based storage strategy. The physical saving is less than it was on a population size of 6 but there was still enough savings made for an extra 6 or 7 generations to be performed.

As Hybrid GAs consist of a GA combined with a secondary method, they can be improved by using an intelligent fitness function on at least the GA part of the hybrid. Also depending on the secondary method used this too may be improved by equipping it with an intelligent fitness function. This would need to be investigated as the secondary method's search patterns would generally be different from the GA's search pattern and as such it may be more beneficial to use a separate intelligent fitness function for the secondary method.

Chapter 5

Partial Fitness Functions

5.1 Introduction

This chapter introduces the concept of partial fitness functions and examines the effect they can have on a GA's performance. Instead of fully testing every individual in a population each generation, a partial fitness function tests only a part of each individual's fitness. Then a percentage of the population is replaced by new individuals. The new individual's parents are selected based on the known fitness of individuals.

In Section 5.5 the effects that different settings have on a partial fitness function are examined. Both the One Max Problem and the Deceptive Trap Function are used to examine the partial fitness function. The GA Hard Problem is not used as it is difficult to create a partial fitness function for it.

5.2 Partial Fitness in Genetic Programming

Genetic programming [80, 81, 88] uses a similar method to GAs except that instead of evolving parameters for an algorithm an actual program is evolved. When the grammar available to the genetic programming system has the ability to produce programs that use loops and recursion the genetic programming system is unable to decide if a program that is having its fitness value tested will finish or if it contains an infinite loop within its code. Koza's method for dealing with the possibility of long/infinite running programs is to specify a maximum length of time that each program can be running for and if a program fails to complete in this time limit then it is assigned a very low fitness score [80].

An alternative method that does not time out programs directly has been provided by Maxwell [97]. If at the end of the specified time a program has failed to finish it is assigned a partial fitness based on the output of the program up to that point. If the program makes it through to the next generation of programs then it is given the chance to continue from where it was stopped. Maxwell claims that his method can require "less effort" and produce solutions that have "greater efficiency".

5.3 Concept of Partial Fitness Functions in Genetic Algorithms

In a standard GA each round (usually referred to as a generation) consists of completely testing the fitness of each individual in the population. The next generation is then produced based on the fitness level of the individuals, the greater the fitness the more likely the individual will be used as a parent for the next generation.

In a GA using a partial fitness function each round (referred to as a partial generation) consists of partially testing the fitness of each individual in the population. Then each individual has its full fitness level estimated by dividing its partial fitness value by the maximum possible fitness value for the amount of the individual's fitness checked. The estimated fitness levels are then used to produce some new individuals to replace those with the lowest estimated fitness in the population. The higher an individual's estimated fitness the more likely it is to be used as a parent for the new individuals. The number of individuals to be replaced after each round is one of the parameters of the GA.

The next round of a GA using a partial fitness function is the same as the previous one except any individual that has had part of its fitness assessed by the partial fitness function has the next part of it assessed and the total fitness of the individual is updated with the new information. The full fitness levels of the individuals are estimated again, the new individuals are created and replace the individuals with the lowest estimated fitness levels.

The motivation behind the design of the partial fitness function is that a standard GA will spend as much time on a poor individual as on a good individual. A partial fitness function is designed to give the GA a chance to quickly look at all the individuals and then decide which ones are worth testing more to get a more accurate fitness value. This should help reduce the time spent by a GA on individuals which after being tested are unlikely to be selected for parents, while the more promising individuals still have their full fitness tested. This technique is similar to the razoring technique [8] used for pruning search trees in game playing problems.

A GA using a partial fitness function differs to a GA using a functional approximation in that the GA using a partial fitness function uses a number of approximation functions to refine an individuals fitness value. By using a number of approximations to assess an individuals fitness the GA gets more accurate information than a GA using a functional approximation but still has the chance to discard poor individuals quickly.

Partial fitness functions are a closer model of evolution than the standard GA model. With the standard GA individuals only get the chance to reproduce at the end of their life. In the real world individuals have the chance to reproduce at different stages of their life. The first chances an individual gets to reproduce are at the early stages of its life when only some of its attributes are known. As an individual gets older then more of its attributes will be known, but it still has to compete with the newer younger individuals to reproduce.

5.4 Partial Fitness Levels

The partial fitness levels are based upon the premise that individuals that start out with a poor level of fitness tend to finish with a poor level of fitness. In effect the GA using a partial fitness function is making decisions based on the *promise* of an individual rather than the absolute value of its fitness. This enables the GA to spend the time it would have spent on an unpromising individual on an individual with more promise.

The partial fitness levels are calculated by splitting the fitness assessment into a number of stages and then for each partial generation one stage in each individual is assessed. The order of the assessment of the stages is fixed as this will add structure to the GA and also allow the user to easily know which parts of each individual's fitness have been assessed.

5.4.1 Comparing Partial Fitness Levels to the Full Fitness Level

A standard GA was run 50 times for both the One Max Problem and the Deceptive Trap Functions. Each of the 50 runs used a different seed for the pseudo random number generator. The GA tackling the One Max Problem had a genome length of 1000 and the GA tackling the Deceptive Trap Functions had a genome length of 80. Both GAs used a population size of 6 and an elitism level of 1 individual.

Each time the GA fitness tested an individual the different partial fitness levels were also calculated for that individual. The One Max Problem's fitness function was split into 10 stages ranging from $\frac{1}{10}$ of the individual tested and increasing by $\frac{1}{10}$ for each stage.

As the GA tackling the Deceptive Trap Functions uses only 80 genes per individual the fitness test was split into 4 stages, $\frac{1}{4}$, $\frac{2}{4}$, $\frac{3}{4}$ and all of the individual tested. It is possible to split it into similar stages as the One Max Problem but then only 8 genes of each individual would be tested per partial generation. Using stages of $\frac{1}{4}$ for the Deceptive Trap Functions results in 20 genes of each individual being tested per partial generation.

Results of the Partial Fitness and Full Fitness Level Experiments

Both the graphs shown in Figures 5.1 and 5.2 show the average partial fitness levels and the full fitness level for all individuals tested at a specific generation. While the graphs shown in Figures 5.3 and 5.4 show the average absolute difference between the full fitness level and the partial fitness level.

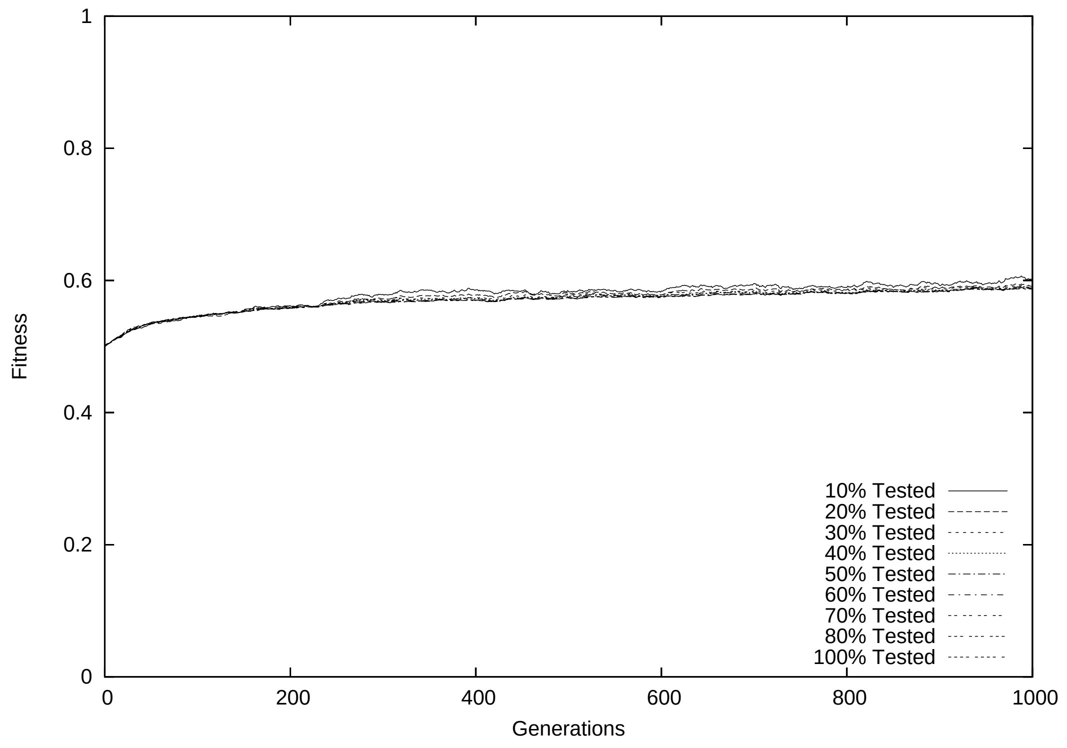

Figure 5.1: The results of the estimation of the individuals tackling the One Max Problem

Figure 5.1 shows that on average for the One Max Problem the partial fitness levels follow the full fitness level very closely. The reason the GA does not appear to have performed as well as previously is because all individuals for each generation are being averaged rather than just the best individual from each generation of each run. The overall average is shown to show on average how close the partial fitness level is to the full fitness level.

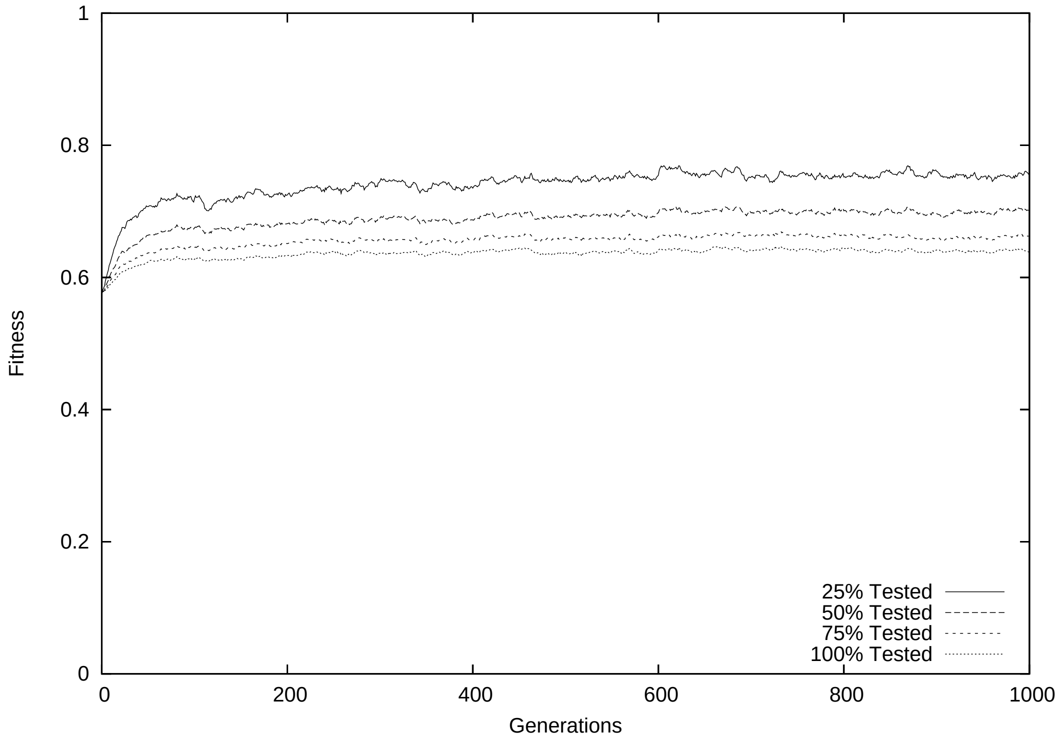

Figure 5.2: The results of the estimation of the individuals tackling the Deceptive Trap Functions

The graph in Figure 5.2 shows that for the Deceptive Trap Functions there is a tendency for the partial fitness function to over estimate an individual's fitness. This make the GA slightly biased towards the new individuals than the older individuals.

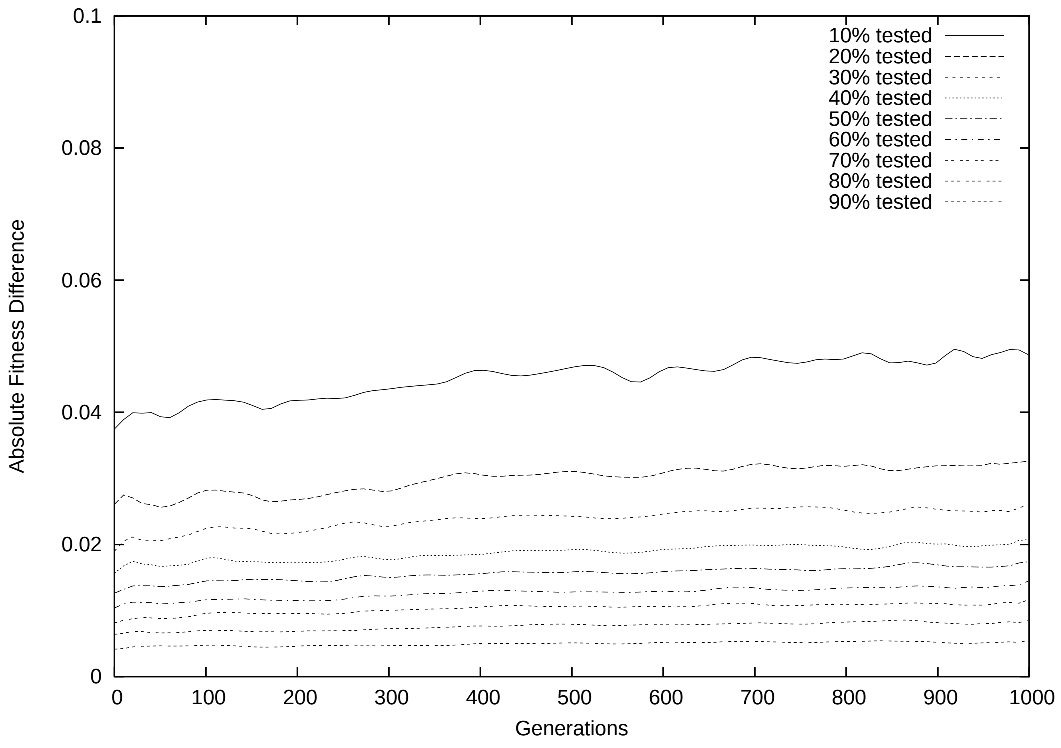

Figure 5.3: The average absolute difference for the estimation of the individuals tackling the One Max Problem

The graph in Figure 5.3 shows that the more of an individual tested by the partial fitness function the more accurate it becomes. With a simple problem like the One Max Problem even testing $\frac{1}{10}$ of an individual produces a reasonably accurate result.

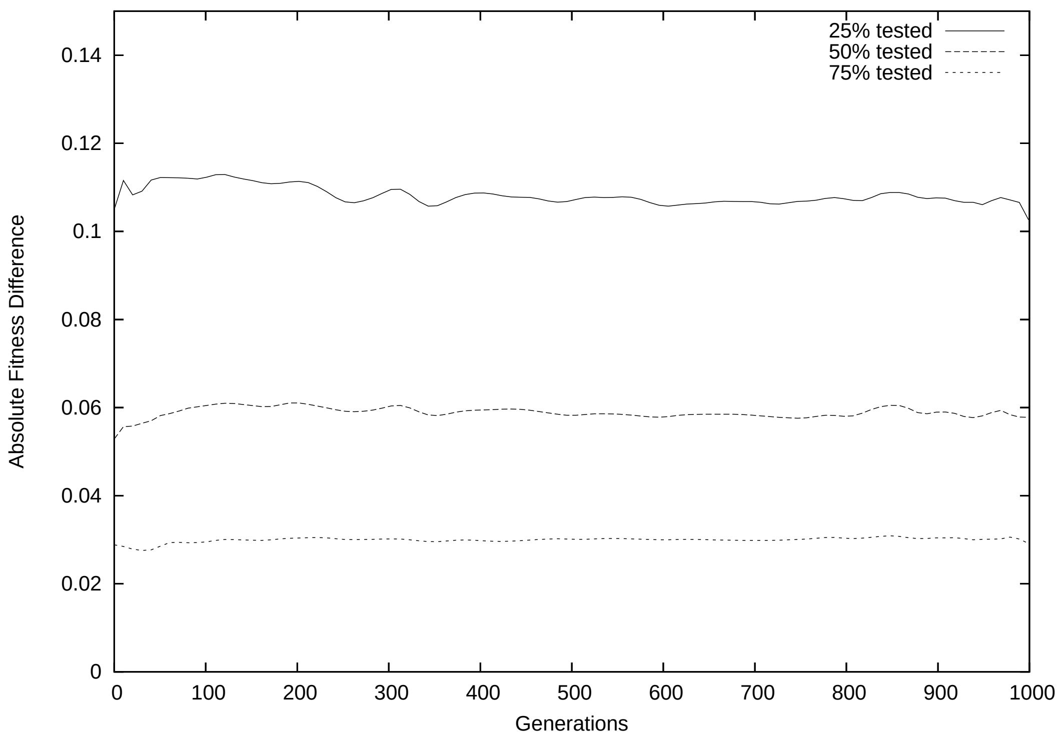

Figure 5.4: The average absolute difference for the estimation of the individuals tackling the Deceptive Trap Problem

The graph in Figure 5.4 shows that on average the lower level of individuals tested ($\frac{1}{4}$) is inaccurate by over 10%. As the amount of the individual tested increases so too does the accuracy. Provided that the number of individuals to be replaced each partial generation is set low enough then this inaccuracy should not cause a problem. The GA equipped with a partial fitness function will increase the accuracy of each individual's partial fitness level in each partial generation.

5.5 Partial Fitness Function Experiments

A GA with a partial fitness function was used with a population size of 60. After each partial test a percentage of individuals with the lowest fitness was replaced with new individuals produced by crossover and mutation. The number of individuals replaced was one of the variables being assessed in the test. The mutation rate was $\frac{1}{n}$ and single point crossover was used.

Two parameters were tested to see what effect they had on the results of the GA with a partial fitness function. The two parameters tested were the amount of an individual tested before the new individuals are generated, and the number of new individuals introduced at each generation. The GA was run against the One Max Problem and the Deceptive Trap Functions. The GA Hard Problem was not used as a test as an individual's fitness is calculated from just one equation. This does not give an option of using a partial fitness function as until the equation is calculated there is no fitness level and once the equation has been calculated the actual fitness level is known.

Both GAs used the same method for estimating the fitness levels of individuals as shown in Section 5.4.

5.5.1 One Max Problem

The GA tackling the One Max Problem was tested 50 times and averages produced for each of the following settings. The same 50 seeds were used to set the pseudo random number generator for each of the settings tested.

- From $\frac{1}{10}$ to $\frac{9}{10}$ of an individual tested. In increments of $\frac{1}{10}$.

- From $\frac{1}{10}$ to $\frac{9}{10}$ of the population replaced at each generation. In increments of $\frac{1}{10}$.

5.5.2 Deceptive Trap Function

The GA using the Deceptive Trap Function was tested 50 times, with averages produced, for each of the following settings. The same 50 seeds were used to set the pseudo random number generator for each of the settings tested.

- From $\frac{1}{4}$ to $\frac{3}{4}$ of an individual tested. In increments of $\frac{1}{4}$.
- From $\frac{1}{10}$ to $\frac{9}{10}$ of the population replaced at each generation. In increments of $\frac{1}{10}$.

5.6 Results

Sections 5.6.1 and 5.6.2 show a summary of the results obtained from the experiments. For the complete set of results from the experiments see appendix C.

5.6.1 One Max Problem

The following Figures (5.5 – 5.8) show the effects of increasing the percentage of the population being replaced after each partial generation on the GA tackling the One Max Problem.

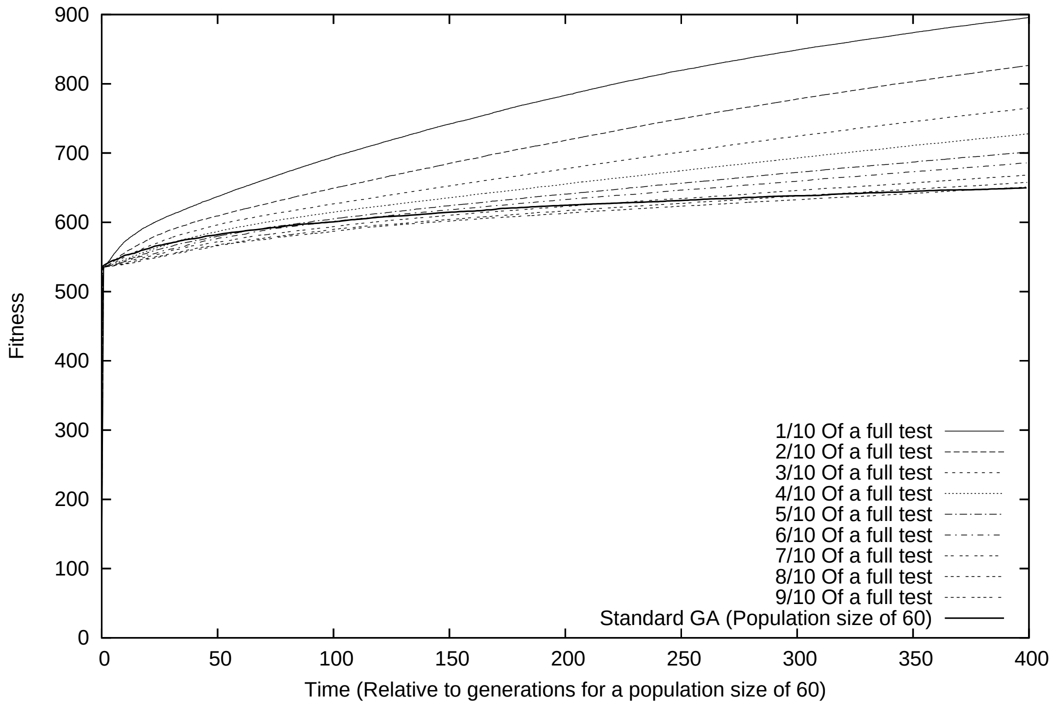

Figure 5.5: Results of using a partial fitness function on the One Max Problem, replacing $\frac{1}{10}$ the population after each partial fitness test

The graph in Figure 5.5 shows that replacing $\frac{1}{10}$ of the population after each partial generation results in the best partial fitness function being the one that tests $\frac{1}{10}$ of an individual each partial generation. The next best partial fitness function is the one that tests $\frac{2}{10}$ of an individual each generation. This pattern continues with the next best being a partial fitness function that tests $\frac{3}{10}$ of an individual each generation, and then one that tests $\frac{4}{10}$ of an individual each generation. The partial fitness function that tests $\frac{9}{10}$ of an individual each generation gives very similar results to a standard GA.

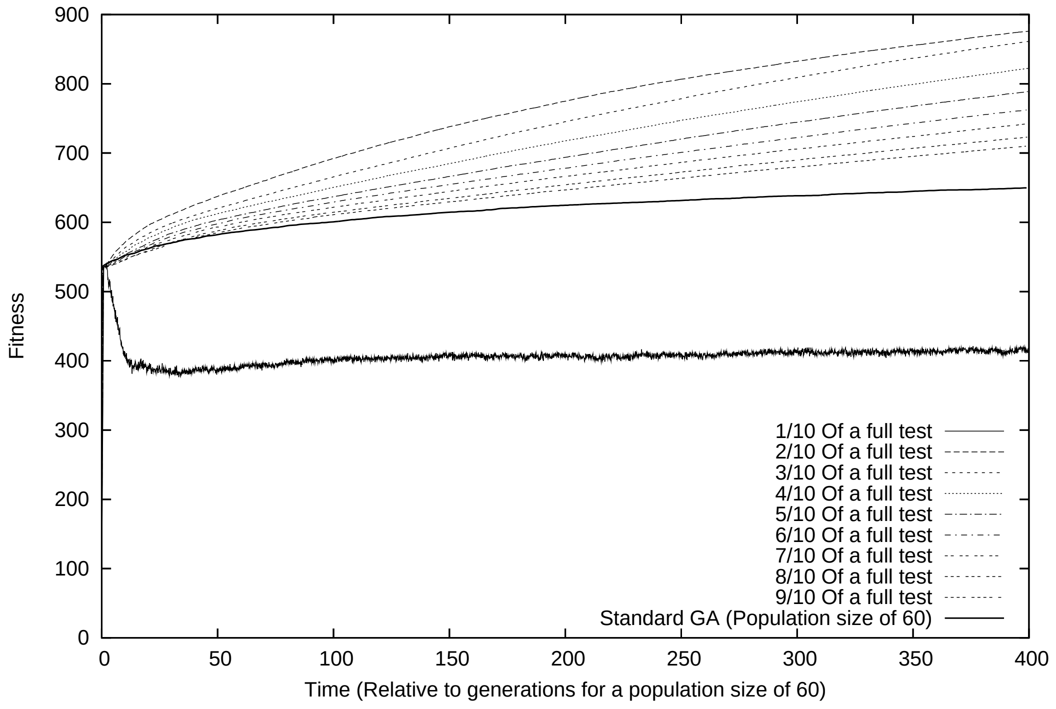

Figure 5.6: Results of using a partial fitness function on the One Max Problem, replacing $\frac{2}{10}$ the population after each partial fitness test

The graph in Figure 5.6 shows that replacing $\frac{2}{10}$ of the population after each partial generation the fitness levels produced by the partial fitness functions testing $\frac{2}{10}$ to $\frac{9}{10}$ of an individual each generation, is greater than for the same fitness functions when replacing $\frac{1}{10}$ of the population. The results of the partial fitness function that tests $\frac{1}{10}$ of an individual each generation performs very badly.

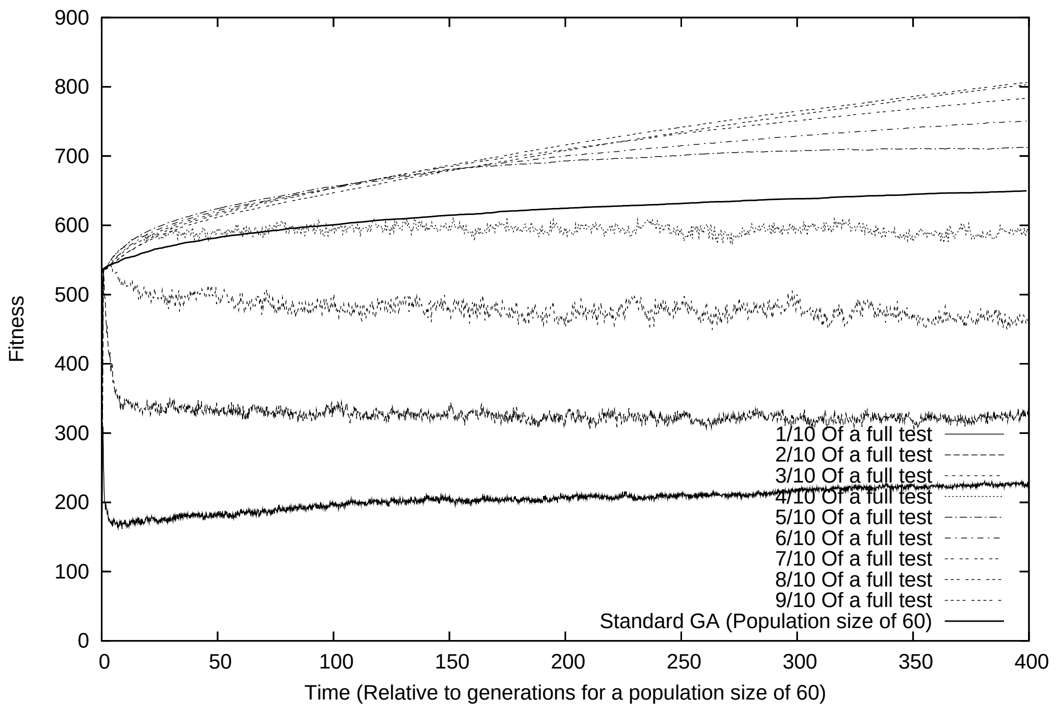

Figure 5.7: Results of using a partial fitness function on the One Max Problem, replacing $\frac{5}{10}$ the population after each partial fitness test

The graph in Figure 5.7 shows that replacing $\frac{1}{2}$ of the population after each partial generation results in the fitness levels, produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, is lower than for the same fitness functions when replacing $\frac{4}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

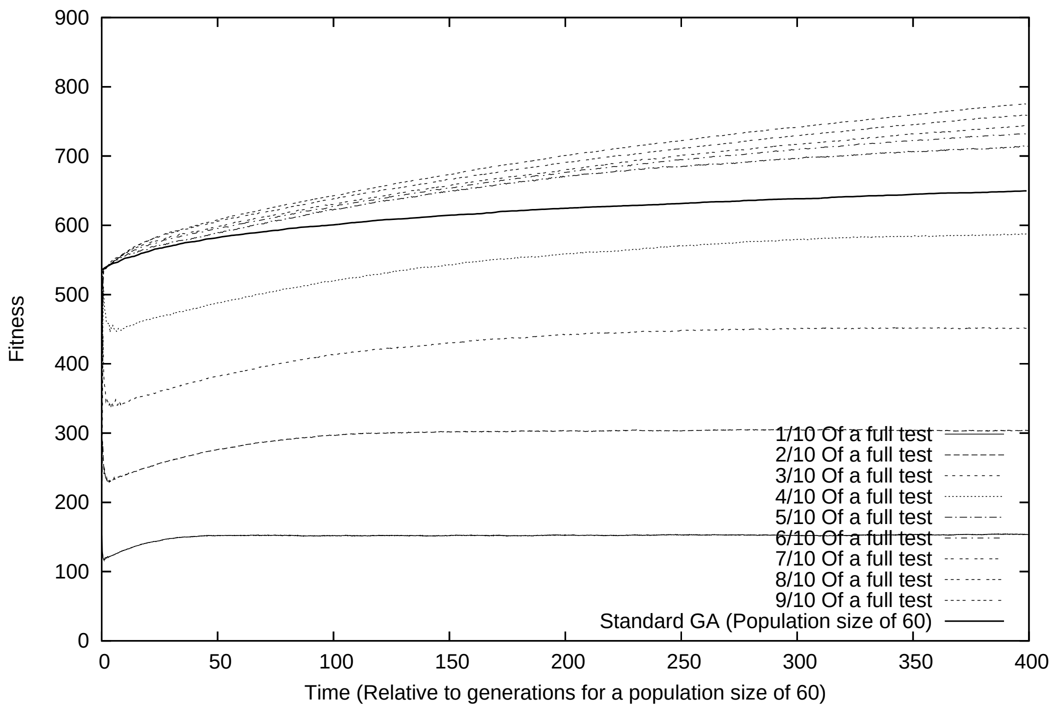

Figure 5.8: Results of using a partial fitness function on the One Max Problem, replacing $\frac{9}{10}$ the population after each partial fitness test

The graph in Figure 5.8 shows that replacing $\frac{9}{10}$ of the population after each partial generation results in the fitness levels, produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, performing slightly worse than for the same fitness functions when replacing $\frac{8}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

Summary of the One Max Problem Results

The graphs in Figures 5.5 – 5.8 show that partial fitness functions can improve the performance of a GA tackling the One Max Problem. The biggest improvement came from replacing $\frac{1}{10}$ of the population after a partial generation that tests $\frac{1}{10}$ of each individual. The graphs also show that partial fitness functions testing less than $\frac{1}{2}$ of each individual only perform well as long as the percentage of the population being replaced is less than or equal to the percentage of each individual being tested per partial generation. This is discussed in the summary of this chapter.

5.6.2 Deceptive Trap Function

The following Figures (5.9 – 5.12) show the effects of increasing the percentage of the population being replaced after each partial generation on a GA tackling the Deceptive Trap Functions.

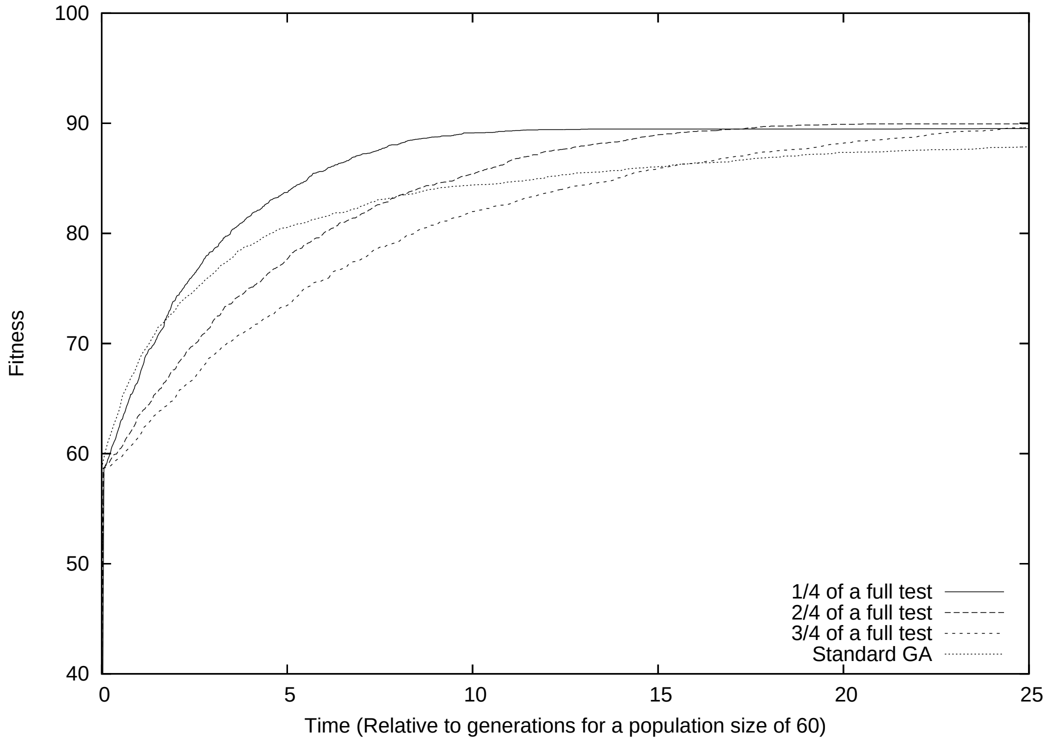

Figure 5.9: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{1}{10}$ the population after each partial fitness tests

The graph in Figure 5.9 shows that replacing $\frac{1}{10}$ of the population after each partial generation results in the fitness levels, produced by the partial fitness functions testing $\frac{1}{4}$ to $\frac{3}{4}$ of an individual, being better than the fitness level produced by a standard GA on the same problem.

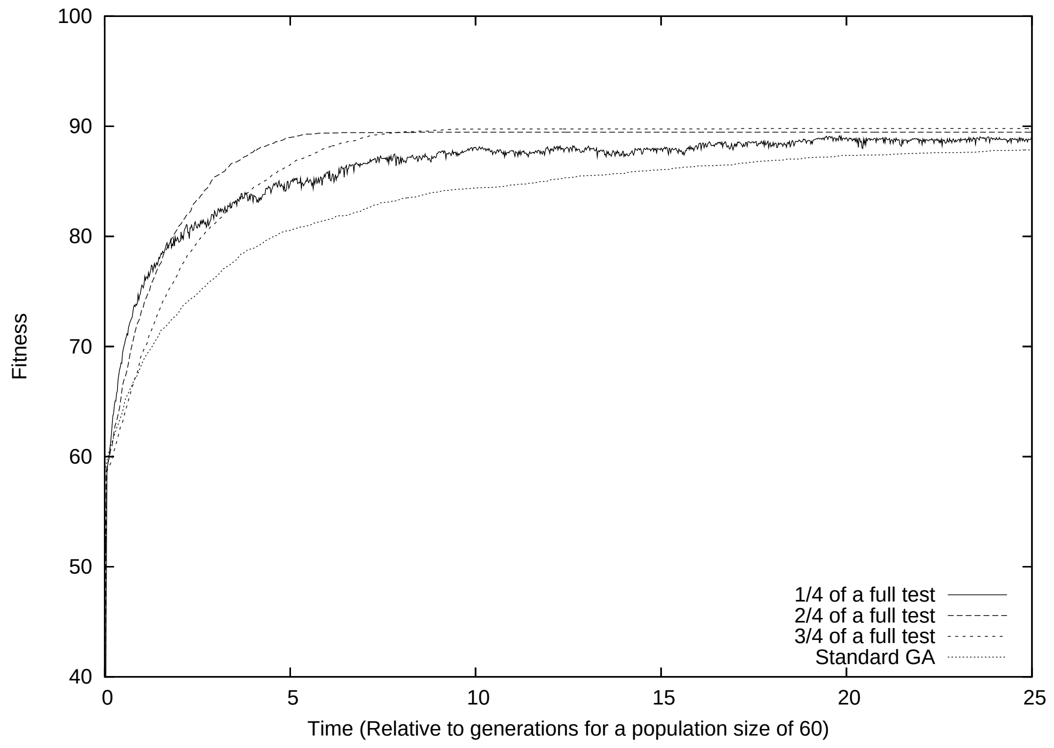

Figure 5.10: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{4}{10}$ the population after each partial fitness tests

Figure 5.10 shows that the replacement of $\frac{4}{10}$ of the population after each partial generation results in the fitness level produced by the partial fitness function testing $\frac{1}{4}$ of an individual has not only dropped slightly but has also started to become unstable. The partial fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ produce very similar results to the previous graph.

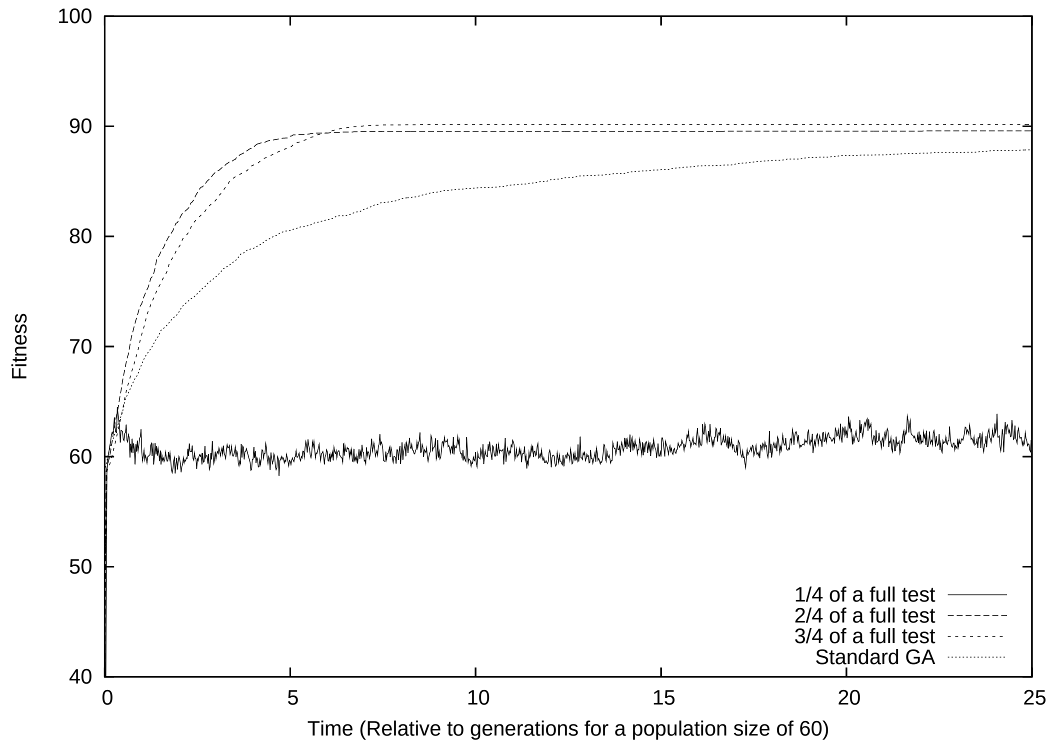

Figure 5.11: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{5}{10}$ the population after each partial fitness tests

The graph in Figure 5.11 shows that replacing $\frac{5}{10}$ of the population after each partial generation results in the fitness level, produced by the partial fitness function testing $\frac{1}{4}$ of an individual each partial generation, dropping well below the level achieved by the standard GA and becoming even more unstable. The fitness levels achieved by the partial fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ are very similar to the previous graph.

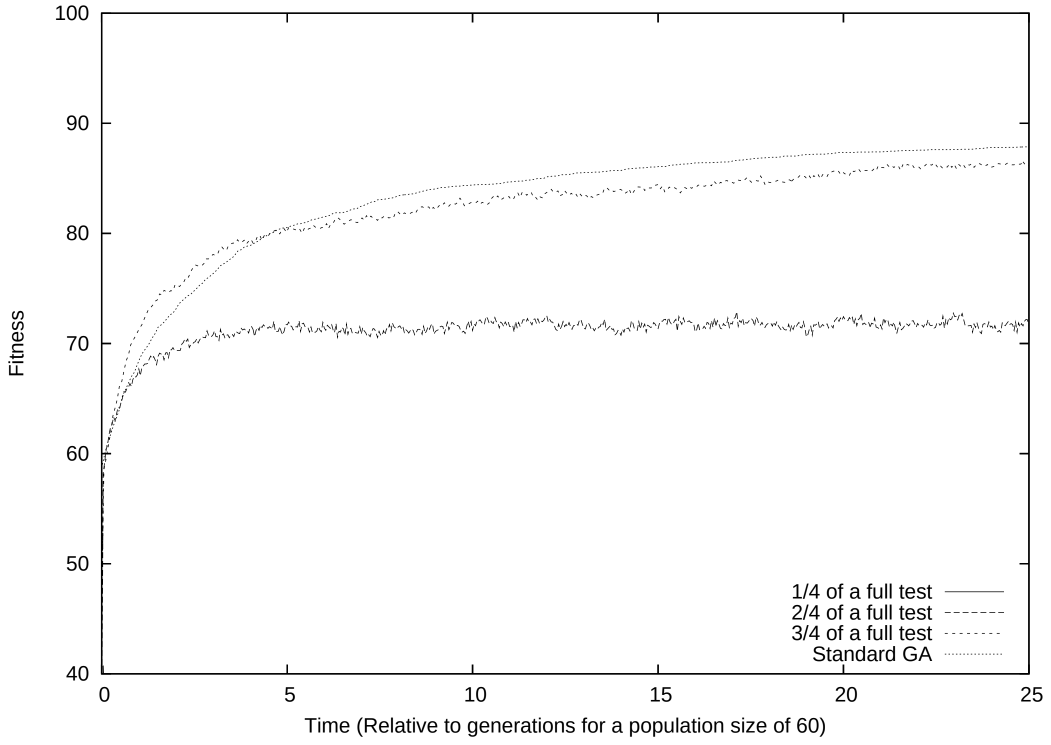

Figure 5.12: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{9}{10}$ the population after each partial fitness tests

The graph in Figure 5.12 shows that the replacement of $\frac{9}{10}$ of the population after each partial generation produces unstable fitness levels for all three of the partial fitness functions tested. (Note that the fitness level for the partial fitness function testing $\frac{1}{4}$ of an individual drops below 40 almost immediately. See Figure C.18 in Appendix C for a full plot of this graph.)

Summary of the Deceptive Trap Functions Results

The graphs in Figures 5.9 – 5.12 show similar results to those of the One Max Problem. As the percentage of individuals being replaced, after each partial generation, increases the more unstable the partial fitness functions testing a lower percentage of each individual becomes.

5.7 Partial Fitness Functions Limitations

To be able to use a GA with a partial fitness function on a specific problem the problem must be capable of being broken up into a number of stages. At the end of each of these stages the known fitness of the individual needs to give a reasonable indication of the overall fitness of an individual.

As the length of runtime available to the GA increased the difference between the partial fitness function and the standard fitness function decreased. This suggests that the shorter the length of time (relative to the length of time taken to calculate a full fitness function) available for the GA the greater the difference between the final result of a normal GA and the final result of a GA equipped with a partial fitness function. The opposite is also suggested that the longer the length of time available the lower the difference between the final result produced by a normal GA and the final result produced by a GA equipped with a partial fitness function.

5.8 Summary and Conclusions

Despite the results of the estimation shown in Section 5.4, which show that the estimation was less than 90% accurate for the Deceptive Trap Functinons and biased towards newer individuals, a GA equiped with a partial fitness function still performed better than a standard GA. The information from this estimate enables the GA equipped with a partial fitness function to easily decide which individuals are promising and which are best to be replaced.

Both sets of results show that partial fitness functions improve the performance of a GA. On all the tests carried out the partial fitness functions only improved performance when the amount of the population being replaced each generation was a lower percentage than the amount of the fitness test being carried out. So when using a partial fitness function it is important to only replace, at most, the same percentage of the population as the percentage of an individual tested each generation.

The problem with partial fitness functions becoming unstable, when the percentage of the population being replaced after each partial generation is greater than the percentage of each individual being tested per partial generation, is caused by the following.

As the number of individuals replaced each partial generation increases then the number of promising individuals that will have more of their fitness tested decreases. The lower the number of promising individuals selected the greater the need for the estimate, of the fitness of those individuals, to be accurate. To improve the accuracy of the estimate more of the individuals fitness must be tested. This is why as the percentage of the population being replaced increases so too does the percentage of each individual being tested, each partial generation, to keep the GA stable.

Partial fitness functions are shown to be problem-dependent as the lower level of fitness testing performed best on the simpler One Max Problem but the higher level of fitness testing performed better on the Deceptive Trap Problem. Also partial fitness functions may not be a viable option for some fitness tests, like the GA Hard Problem, where there is no partial stage in the fitness test that can be worked from. Partial fitness functions have also been shown to be dependent on the available time. The more limited the time then the lower the level of fitness testing that should be used. On both problems the lowest tested level of fitness checking performed best over the initial period when the replacement level is set correctly.

Chapter 6

Use of Intelligent Fitness Functions and Partial Fitness Functions on a Monoalphabetic Substitution Cipher

6.1 Introduction

In this chapter Intelligent Fitness Functions and Partial Fitness Functions are applied to a GA designed to help break monoalphabetic substitution ciphers. Both the GA using an Intelligent Fitness Function and the GA using a Partial Fitness Function are tested against the standard GA with the same configuration (Population Size, Random Seed, etc.)

The fitness function for a GA tackling a monoalphabetic substitution cipher has to produce a collection of frequency counts for individual characters, digrams and trigrams and compare these against an expected frequency count. The fitness function therefore takes a lot longer to run than those that tackled the simple test problems examined in the previous chapters. This problem is better suited to assessing intelligent fitness functions and partial fitness functions than the packet transmission problem that motivated the work (described in chapter 2) for a number of reasons.

- A known optimal solution to the problem enables us to gauge the effects the new methods have on GAs.
- Results are not dependent on unknown factors and as such are repeatable (e.g. for a GA tackling the packet transmission problem the network load may have an effect on the results.)
- As monoalphabetic substitution ciphers have been solved by GAs in the past [58] there should be an improvement in the fitness levels of the GA as it evolves. The packet transmission problem has not been solved by GAs at the time of writing and as such it is unknown if there would be an improvement in the fitness levels of the GA.

Nevertheless the improvements demonstrated if transferred to the packet transmission problem would result in substantial savings.

6.2 GAs Use in Cryptanalysis

GAs have been used to help cryptanalysts in breaking a small number of ciphers. Monoalphabetic substitution ciphers are one of the easiest to tackle and as such they have been targeted as a good place to start [58]. Others have had success applying GAs to breaking more complex ciphers, including rotor based ciphers [4] (like the Enigma machine used by the Germans in the second world war [25]). Cryptography is a continually advancing science and GAs have not been successfully used to tackle more modern ciphers like DES [36] and RSA [131].

6.3 Monoalphabetic Substitution Ciphers

Monoalphabetic substitution is one of the oldest forms of enciphering text. It consists of using a constant one to one mapping of characters to substitute each character of the original message (known as the plaintext) with the relevant character from the mapping (known as the key). Once each character in the plaintext has been substituted the encoding is over and the resulting message (known as the ciphertext) can be sent to its destination. At the message's destination the process is reversed to obtain the plaintext. This type of cipher is very old and has been used by many famous people in history. [136, 74]

Example The following is an example of using a monoalphabetic substitution cipher. To encrypt the message, "THIS IS AN EXAMPLE", using the key shown in table 6.1 the following process is followed. The first character in the plaintext is taken, "T", and looked up in the key. The first letter in the ciphertext is then the one shown in the key to replace “T”, which in this case is “G”. Then the second letter is taken, “H” and looked up in the key. The second letter in the ciphertext is then the one shown in the key to replace “H”, which is “S”. This process is repeated for every letter in the plaintext, which results in a ciphertext of “GSRH RH ZM VCZNKOV”.

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M
Ciphertext	Z	Y	X	W	V	U	T	S	R	Q	P	O	N
Plaintext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	M	L	K	J	I	H	G	F	E	D	C	B	A

Table 6.1: An example key for a substitution cipher

To decrypt the ciphertext the recipient of the message reverses the process. So the first letter of the ciphertext is taken, “G” this is looked up in the ciphertext side of the recipients copy of the key and replaced with the related letter on the plaintext side, “T”. Then the second letter of the ciphertext is taken, “S”, and replaced with the related letter on the plaintext side of the key, “H”. Once this has been done for each letter in the ciphertext the resulting message is shown “THIS IS AN EXAMPLE”.

Frequency Analysis

The traditional method for breaking a monoalphabetic substitution cipher is through the use of frequency analysis. This method was first documented by the Arab scientist Abū Yūsūf Ya‘qūb ibn Is-hāq ibn as-Sabbāh ibn ‘omrān Ismaīal-Kindī[2, 1]. Despite Al-Kindī’s work containing detailed analysis of statistics and Arabic syntax and phonetics he summarised his cryptanalysis method in two paragraphs.

“One way to solve an encrypted message is, if we know its language, is to find a different plaintext of the same language long enough to fill one sheet or so, and then we count the occurrences of each letter. We call the most frequently occurring letter the ‘first’, the next most occurring letter the ‘second’, the following most occurring letter the ‘third’, and so on, until we account for all the different letters in the plaintext sample.

Then we look at the ciphertext we want to solve and we also classify its symbols. We find the most occurring symbol and change it to the form of the ‘first’ letter, the next most common symbol is changed to the form of the ‘second’ letter and the following most common symbol is changed to the form of the ‘third’ letter, and so on, until we account for all symbols of the cryptogram we want to solve.”

Example If we apply Al-Kindī’s method to the ciphertext produced in our first example then we get the following. The ciphertext was “GSRH RH ZM VCZNKOV” and the frequency of letters in this ciphertext is shown in table 6.2.

Letter	Count	Letter	Count
H	2	R	2
V	2	Z	2
G	1	S	1
M	1	C	1
N	1	K	1
O	1		

Table 6.2: Frequency of Characters in the Ciphertext

The most common letters in English are the letters “E”, “T”, “A”, “O” and “I”. So in the key used, these letters are more likely to map to “H”, “R”, “V” and “Z”. To help decide which letter is the most likely to represent the common letters in the ciphertext, the layout of the ciphertext can be examined. In the ciphertext the letter “R” is always followed by the letter “H” and this pair of letters also exist as a single word on their own as well as the end of the first word. This suggests that “R” maps to “I” and “H” maps to “S” as the pair “IS” is most common to appear in this pattern.

The message now looks like this “GSis is ZM VCZNKOV”, this suggests that the letter “G” maps to the letter “T” while the letter “S” maps to the letter “H”. This leaves the letters “E”, “A” and “O” as probable choices for the last two common letters in the ciphertext “Z” and “E”. “Z” is chosen to map to the letter “A” as the letter “V” is likely to map to a consonant and hence the last word should be preceded by the word “an“, this also gives the mapping of the letter “M” to “N”.

The message now looks like “This is an VCaNKOV”. The letter “E” is chosen as a mapping to the letter “V”. The cryptanalyst is now left looking for a word seven letters in length, that starts and ends with the letter “E” and has “A” as the third letter. The cryptanalyst would not have much difficulty in guessing the last word as “example”. In fact a search of a spell checker’s dictionary reveals the following 6 possibilities of which “Example” is the only one which fits with the rest of the known text both syntactically and semantically.

- Erasure
- Emanate
- Evacuee
- Evasive

- Examine
- Example

6.3.1 GA to Solve a Monoalphabetic Substitution Cipher

A GA was designed to help in the breaking of a monoalphabetic substitution cipher. Each individual in the GA consisted of a sequence of letters (A – Z). Each letter could only appear once in each individual and each individual must contain all twenty-six letters, the only difference between individuals is the ordering of the twenty-six letters.

The GA uses real encoding rather than binary encoding as this allows the GA to use crossover and mutation operators that are easier to implement and will always produce valid individuals. As the intelligent fitness functions and partial fitness functions operations are independent of the encoding method used for the genome they can be applied to this GA using real encoding in the same way that they would be applied to a binary encoded GA.

The crossover operator uses a variation on the single point crossover method. It picks a random point in the genome as the crossover point. The first part of the genome, up to the crossover point, is then carried forward to the child genome. A copy of the second parent is made and as each letter up to the crossover point is copied forwards it is removed from the copy of the second parent. After all the letters up to the crossover point have been carried forward then letters left in the copy of the second parent are appended to the end of the child's genome.

Example The following example shows the crossover operator in action. Table 6.3 shows the selected parents, in this example the parents only consist of 5 genes. The crossover point is picked as between the third and fourth gene. So the first gene of parent one, "A", is carried forwards as the first gene of the child. After carrying the gene forwards to the child the matching gene from the copy of the second parent is removed, in this case it is also the first gene. Then the second gene of parent one, "B" is carried forwards to the child. After carrying the gene forward the matching gene in the copy of parent two is removed, this is the fourth gene. The third gene, "C", is then carried forwards from parent one to the child. After carrying the third gene forward the matching gene from the copy of the second parent is then removed, the second gene this time.

Table 6.4 shows the state of the first parent, the copy of the second parent and the child after the crossover point is reached. At this stage the first gene still contained in the copy of the second parent, "E", is appended to the child's genome. Then finally the last gene contained in the copy of the second parent, "D" is appended to the child's genome. This results in the new individual shown in table 6.5.

Parent 1	A	B	C	D	E
Parent 2	A	C	E	B	D

Table 6.3: Parents

Parent 1	A	B	C	D	E
Copy of Parent 2			E		D
Child	A	B	C		

Table 6.4: After First Stage of Crossover

Child	A	B	C	E	D

Table 6.5: After Crossover

The mutation operator swaps two randomly chosen genes over. As the GA is using real encoding this means that two letters in the key ordering are swapped over. Both the crossover and mutation method always produce valid individuals provided they are supplied with valid individuals.

To fitness test each individual the GA uses the individual's genome as the key to decipher the message. Once deciphered the resulting plaintext has a frequency table produced for it. This table is then compared to a frequency table generated from a collection of documents written by the same author. The frequency table consists of not only the frequency of single letters but also the frequency of pairs of letters and triplets of letters.

Each individual also scored extra fitness if the first few characters from their resulting plaintext matched a crib. A crib is a known part of a message, usually the first part of a message can be known to an attacker, e.g. if the sender always starts their letters with the word "Dear" then this would be a good crib.

Limitations

The GA can be applied to any monoalphabetic substitution cipher that uses an English alphabet, changes to the GA would need to be made for it to break messages whose plain text is in another language. The genome would need to be changed to take into account a different alphabet. Also the frequency tables used by the fitness function would need to be modified as different languages have different standard frequency counts.

6.4 Intelligent Fitness Functions

The GA described in Section 6.3.1 is compared to a GA equipped with an intelligent fitness function. Both GAs were configured based on the results of research described in Chapters 3 and 4 using a population size of 6 and an elitism level of 1. The intelligent fitness function was equipped with a long term memory with a size of 30 individuals. The storage strategy used by the long term memory was time based.

The periods of time reported in Section 6.4.1 for the tests were produced on a machine with the following specification.

Processor	Athlon XP 1500+ processor
Memory	364MBs.
Operating System	Debian Linux.
Language	C++

6.4.1 Results

The two GAs are compared on three criteria, the length of time taken to run the GA for 10,000 generations, the number of fitness tests actually run and the fitness levels achieved by the GAs against the number of fitness tests performed. The GAs were run 50 times each so an average result could be obtained. Both GAs were run using the same 50 seeds for their pseudo random number generator.

On average after 10,000 generations the standard GA and the GA using an intelligent fitness function both achieved a key that was 85% correct. The GA using an intelligent fitness function achieved this level in 66% of the number of fitness tests as the standard GA. The T-Test results show this difference to be 99.9% significant.

Time

Table 6.6 shows the length of time in seconds each run of the standard GA took. On average each run took 4 minutes and 45 seconds to perform 10,000 generations. Table 6.7 shows the length of time in seconds that the GA, with the intelligent fitness function, took to perform 10,000 generations. The average time for the GA, using an intelligent fitness function, was 3 minutes and 14 seconds.

The GA, using an intelligent fitness function, showed an average saving of 1 minute and 31 seconds over the standard GA. That is a 31.9% saving over the standard GA.

288.67	285.49	284.66	284.30	283.71
284.42	285.26	286.01	285.75	285.53
285.32	285.10	285.05	285.62	285.94
285.77	285.52	285.61	285.42	285.36
285.26	285.38	285.32	285.30	285.45
285.71	285.68	285.76	285.62	285.63
285.64	285.79	285.69	285.69	285.64
285.63	285.56	285.52	285.59	285.61
285.64	285.57	285.51	285.56	285.68
285.70	285.67	285.91	285.92	285.86
Avg	285.541			

Table 6.6: Time of the standard GA runs, in seconds

192.33	195.35	194.85	195.91	195.55
195.17	194.92	194.80	194.44	194.22
194.03	194.33	194.25	194.21	194.11
193.92	194.24	194.18	194.03	194.02
194.08	194.04	194.22	194.15	194.14
194.20	194.26	194.33	194.24	194.27
194.24	194.40	194.33	194.35	194.43
194.41	194.38	194.41	194.35	194.40
194.43	194.42	194.45	194.39	194.44
194.39	194.47	194.50	194.51	194.53
Avg	194.38			

Table 6.7: Time, in seconds, of the runs of the GA using an intelligent fitness function

Fitness Tests Performed

The graph in Figure 6.1 shows the number of fitness tests performed by the two GAs against the number of generations carried out. The GA using an intelligent fitness function has carried out just over 66% of the number of fitness tests carried out by the standard GA by the end of the 10,000 generations.

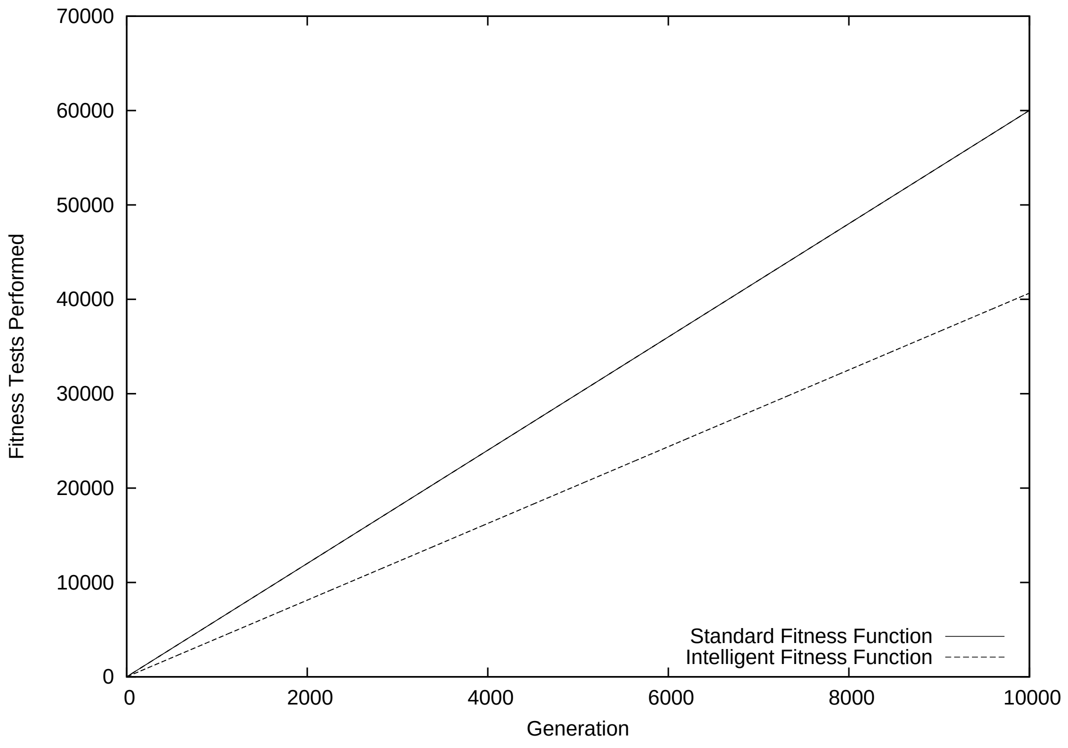

Figure 6.1: Fitness Tests Performed by the standard GA and the GA using an intelligent fitness function.

Fitness Levels

Figure 6.2 shows the fitness levels of both the GAs against the number of fitness tests performed. The results are only shown up to 40,000 fitness tests as this is approximately the average number of fitness tests performed by the GA using an intelligent fitness function. The actual number being 40,649.

The GA using an intelligent fitness function achieved a better fitness level over 40,000 fitness tests than the standard GA.

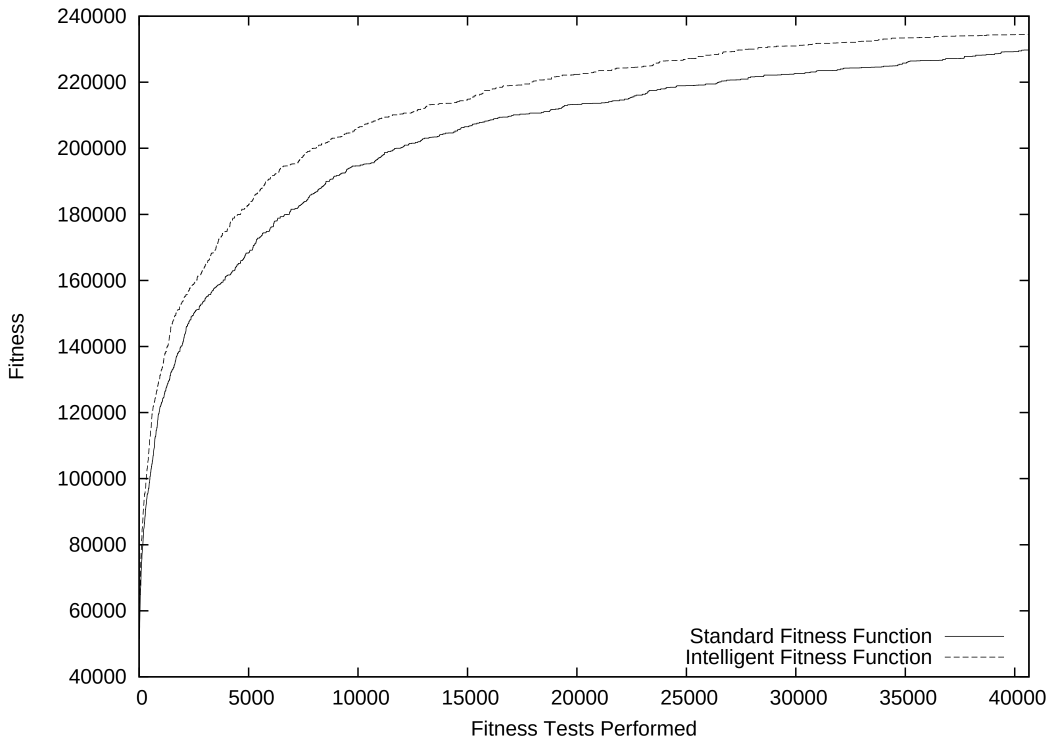

Figure 6.2: Average Fitness Levels of both the GAs against the number of fitness tests performed.

6.5 Partial Fitness Functions

The GA described in Section 6.3.1 is compared to a GA equipped with a partial fitness function. Both GAs have a population size of 60 and are run 50 times for 1000 full generations. The standard GA used elitism with a level of 1. The GA using a partial fitness function tests $\frac{1}{3}$ of each individual per partial generation and replaces $\frac{1}{3}$ of the population each partial generation. The settings for the GA with a partial fitness function are based upon the results of Chapter 5. As $\frac{1}{3}$ of each individual is tested per partial generation there are 3 partial generations to each full generation. Both GAs were started with the same 50 seeds for their pseudo random number generators.

6.5.1 Results

The graph in Figure 6.3 shows the fitness levels achieved by both the standard GA and the GA using a partial fitness function. The GA using a partial fitness function gets off to a better start over the first 500 partial generations. Over the next 2500 partial fitness functions the GA using a partial fitness function keeps the lead in fitness that it gained initially.

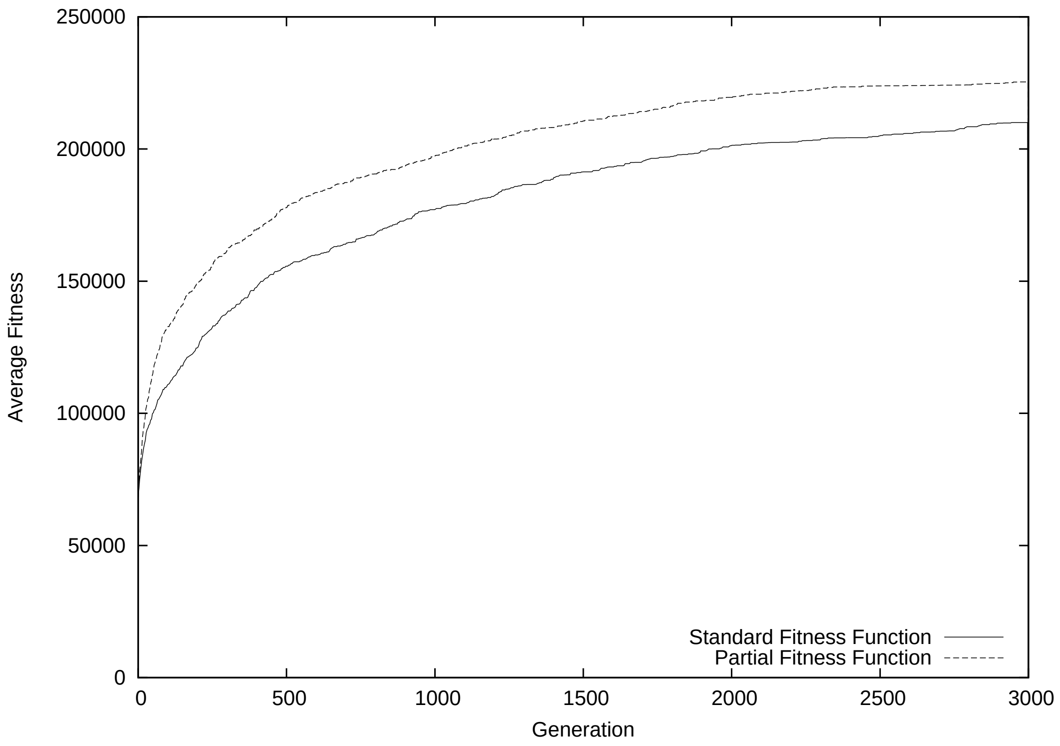

Figure 6.3: Fitness levels, of the standard GA and the GA using a partial fitness function, against number of partial generations.

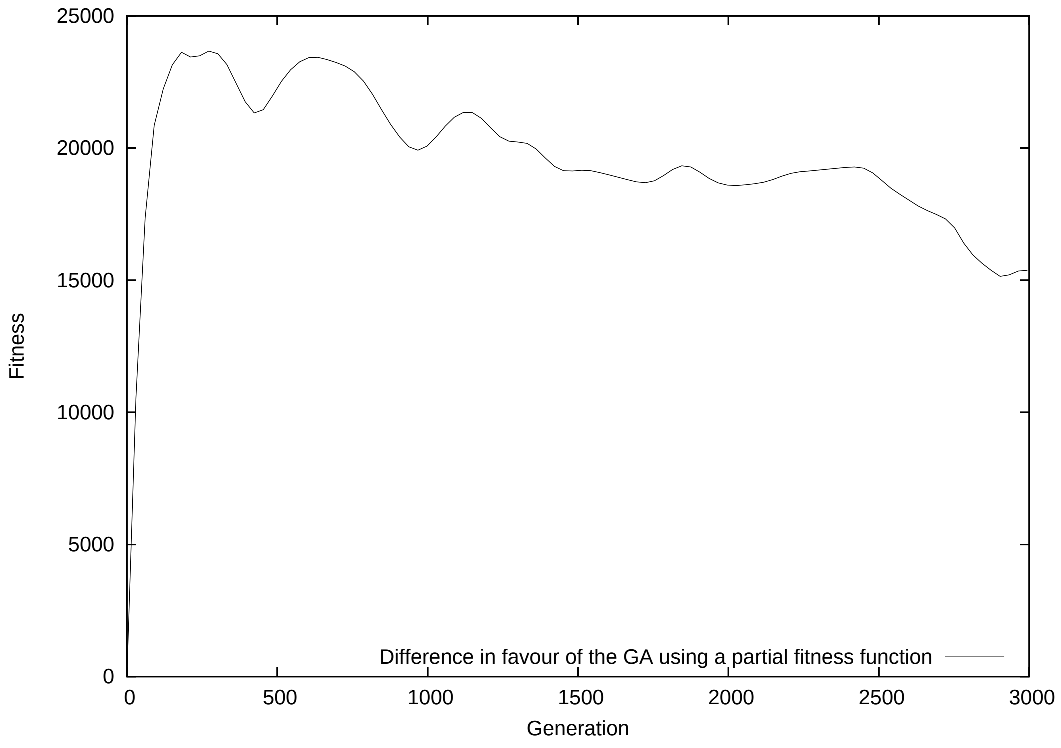

Figure 6.4: Difference between the fitness levels, of the standard GA and the GA using a partial fitness function, against number of partial generations.

The graph in Figure 6.4 shows the difference between the fitness levels of the two GAs. At the early stages the difference between the two GAs increases above 24,000 in favour of the GA using a partial fitness function. As the number of generations increases the GA with a standard fitness function reduces the difference. After 3,000 partial generations (1,000 normal generations) the difference between the two GAs is still 15,000 in favour of the GA using a partial fitness function.

When the two GAs are compared with the amount of the key that is correct then there is very little difference on average (see Table 6.8) and that difference is shown by

	Standard GA	Partial Fitness Function GA
Fitness		
Mean	210039	225450
Standard Deviation	11137	4070
Standard Error	1575	575
Key Correctness		
Mean	17.26	17.46
Standard Deviation	8.46	1.59
Standard Error	1.19	0.22

Table 6.8: Average, Standard Deviation and Standard Error of the best final fitness of each of 50 runs

the T-test result of 0.7 to not be significant (see Table 6.9). The best of all the runs shows the GA with a partial fitness function as 8% better than the the best of all the runs for the GA with a standard fitness function.

It is interesting to note that the average of the runs for both GAs produces keys that are at a very similar level of correctness, yet the GA with a partial fitness function is scoring significantly higher fitness levels. This shows that the GA with a partial fitness function is producing a key that has more important letters in correct places. The important letters being the more common ones as this leaves the more obscure letters for the cryptanalyst to work out, which is easier for a cryptanalyst (It is easy to spot that 'ALMHA' should be 'ALPHA' than 'MLPHM'). Figure 6.5 shows the first 100 deciphered characters for an average key and the best key for both GAs.

	Fitness	Key Correctness
T-Test	0.99	0.7

Table 6.9: T-Test results for fitness levels difference and key correctness difference

```
Standard Fitness Function

  Average Fitness Level Decipherment

    THEWA VERTU MBLER LOCKT HEWAV ERTUM BLERL OCKWA SZENE LOPEZ

    ASALO WCOST LOCKT HATOV VEREZ AREAS OFABL EZEXR EEOVS ECURI


  Best Fitness Level Decipherment

    THEWA FERTU MBLER LOCKT HEWAF ERTUM BLERL OCKWA NDEVE LOPED

    ANALO WCONT LOCKT HATOF FERED AREAN OSABL EDEQR EEOFN ECURI


Partial Fitness Function

  Average Fitness Level Decipherment

    THEWA FERTU MBLER LOCKT HEWAF ERTUM BLERL OCKWA SXEPE LOGEX

    ASALO WCOST LOCKT HATOF FEREX AREAS ONABL EXEQR EEOFS ECURI


  Best Fitness Level Decipherment

    THEWA FERTU MBLER LOCKT HEWAF ERTUM BLERL OCKWA SDEVE LOPED

    ASALO WCOST LOCKT HATOF FERED AREAS OZABL EDEGR EEOFS ECURI
```

Figure 6.5: Examples of text deciphered using keys from the standard GA and the GA using a partial fitness function.

6.6 Summary and Conclusions

The same pattern in the results for the intelligent fitness function, shown in Chapter 4, has been repeated with a GA tackling a monoalphabetic substitution cipher. The fitness test was more computationally intensive than those tested in Chapter 4 and as such the savings on time are more apparent. The intelligent fitness function achieved a saving in time of 31.9%.

The GA using a partial fitness function has shown again the same ability to perform better than a standard GA, as shown originally in Chapter 5. The GA using a partial fitness function achieved on average a level of correctness 1% higher than the standard GA, which is shown by the T-Test result of 0.7 as not being significant. The best run of the partial fitness function achieved a level of correctness 8% better than the best run of the standard GA

Comparing the two GAs fitness levels shows that there is a difference of 15,411 in favour of the partial fitness function which is significant at the 99% level. This means that while the GA with a partial fitness function is only getting the same percentage of the key correct as the normal GA, the parts of the key it does get correct are more important.

While none of the GAs managed to get a 100% correct key in any of the runs, even the worst runs achieved keys that were better than 65% correct, and the best runs achieved keys that were better than 83% correct with some achieving keys that were above 91% correct.

The results in this Chapter show that intelligent fitness functions and partial fitness functions can be used to help GAs being used for cryptanalysis of monoalphabetic substitution ciphers.

Even though no complete solution was found in the limited time given to the GAs, the partial solutions achieved would be useful for a cryptanalyst. After all it is better to have to start with 65% of a solution than no solution. By using the partial solutions along with the frequency counts of the enciphered message the cryptanalyst can easily discover where the frequency counts differ from the expected and quickly work out which parts of the key are wrong and which are correct.

Chapter 7

Discussion and Conclusion

7.1 Introduction

This Chapter discusses the use of intelligent fitness functions and partial fitness functions to improve the efficiency and performance of GAs. Also possible future work and the contribution to knowledge by this Thesis will be discussed.

7.2 Discussion

7.2.1 Intelligent Fitness Functions

Chapters 4 and 6 introduced intelligent fitness functions and showed the degree of savings they can achieve. Intelligent fitness functions can be applied to the following areas as well as to standard GAs.

GAs Using Approximations for Fitness Functions As intelligent fitness functions do not change how the fitness of an individual is calculated they can be applied to GAs which have an approximation as a fitness function in the same way they would be applied to a standard GA. As some approximations can be calculated faster than an intelligent fitness function can search its memory they may not always be suitable. Equation 4.1 should be used to calculate if it would be suitable to use an intelligent fitness function with the approximation.

Hybrid GAs As the GA part of a hybrid GA is usually just a normal GA then it can be equipped with an intelligent fitness function to improve it. This would the same as adding an intelligent fitness function to a normal GA. Again Equation 4.1 should be used to decide whether it would be beneficial to use an intelligent fitness function.

Some of the secondary methods used by hybrid GAs use a fitness function to assess the fitness of neighbouring individuals (e.g. Local Search, Simulated Annealing, etc.). These fitness functions could be replaced with an intelligent fitness function which may lead to an improvement in the efficiency and performance of the hybrid GA. As the secondary method will have a different level of efficiency it would be necessary to investigate the secondary method separately to decide if an intelligent fitness function would be beneficial.

If it is beneficial for the secondary method to use an intelligent fitness function then the next question would be "Should it use the same memory as the GA or should it use a separate memory?" The intuitive answer would be that the search patterns of GAs and secondary methods are different so stored individuals for one would not be useful for the other. If this is the case then it would suggest that separate memories would be better for the GA and the secondary method. This would need to be examined for a variety of secondary methods and problem classes.

Other Heuristics As with some of the secondary methods of hybrid GAs any heuristic that uses a fitness function can have an intelligent fitness function applied to it. Each of these heuristics would have to be investigated to see if it would be worth while equipping it with an intelligent fitness function. Equation 4.1 can be used to calculate if an intelligent fitness function would be worth while for a heuristic. Some heuristics are less likely to benefit from an intelligent fitness function than others. For example a tabu search is less likely to encounter duplicates as it is designed to not move back to locations it has recently searched and as such would probably not benefit much from an intelligent fitness function.

Intelligent fitness functions will not help improve the performance of a GA which falls into one of the following categories.

Quick Fitness Function. If the GA has a fitness function that can be calculated quicker than the intelligent fitness function can search and maintain its memory then the GA's performance will be lowered. Equation 4.1 can be used to decide if a GA will be able to benefit from an intelligent fitness function.

High Efficiency. If the GA has a high level of efficiency already then an intelligent fitness function will be less likely to help improve the performance. Equation 4.1 can again be used to decide if the GA will benefit from an intelligent fitness function.

Changing values of fitness. If the GA is tackling a problem where an individual will not always return the same fitness value each time it is tested then an intelligent fitness function is not suitable as the value it has stored may not be the current fitness value for an individual.

7.2.2 Partial Fitness Functions

Partial fitness functions could be applied to the problem of finding the optimum settings for packet transmission as described in Chapter 2. Each partial generation could last for 2 days. At the end of each partial generation the average delay for each individual would be used. As the fitness of each individual is the average delay there is no need to normalise the fitness value. Following the results in Chapter 5, $\frac{1}{7}$ of the population should be replaced after every partial fitness function to keep the GA stable. If more partial generations were required then the partial generations could last for 1 day and $\frac{1}{14}$ of the population could be replaced.

Hybrid GAs can also benefit from the use of partial fitness functions. The GA part of a hybrid GA could use a partial fitness function provided it meets the same criteria as a normal GA using a partial fitness function. The secondary method will in most cases require an individual's full fitness level as they tend to work on single solutions rather than a population of solutions. Some heuristics that use a population may be able to benefit from a partial fitness function though these would need to be investigated.

Maxwell claimed that his method for genetic programming produced programs that have a "greater efficiency"[97]. In genetic programming a solution that has a greater efficiency than another is of a higher quality. These claims match the results of the partial fitness function tested in Chapter 6 where the solutions produced by the partial fitness functions were the better parts of the key and as such were a higher quality.

Partial Fitness Functions cannot be applied to every GA as some are not suitable. Partial fitness functions cannot be used in the following case.

Single Stage Fitness Functions. If the fitness function cannot be broken up into a number of stages that give a reasonable indication of the overall fitness of an individual then there is no partial fitness levels available to the GA to work with.

Also the results in Chapters 5 and 6 suggest that the larger the number of generations that a GA can run for the lower the effect that a partial fitness function has on the performance of the GA.

7.3 Contribution to Knowledge

This Thesis has addressed the problem of improving the performance and efficiency of GAs. The efficiency has been improved by decreasing the number of duplicate fitness tests being performed. Improving the efficiency in this way reduces the time the GA takes to achieve a number of generations and as such also improves the performance of the GA. The performance has also been improved by increasing the frequency of evolutionary steps without decreasing the population size.

The improvement in efficiency was achieved through the use of intelligent fitness functions. Standard fitness functions can be modified to become intelligent fitness functions without having to alter any other part of the GA. This means that current GAs can be modified to take advantage of intelligent fitness functions without needing to alter the GA system used, only the fitness function. Once a GA is using an intelligent fitness function it can then use a smaller population size without decreasing the efficiency of the GA by much. The decrease in the population size gives an increase in the number of generations available in a fixed length of time. As shown in Chapter 3 this gives an increase in the performance of the GA.

The second improvement has been achieved through the use of partial fitness functions. The partial fitness functions need modifications in the way the GA handles the production of the next generation. This means that an existing GA system would need to be modified to use a partial fitness function. Once the system has been adapted to the partial fitness function the GA can then discard unpromising individuals and select promising individuals without having to fully fitness test each individual.

7.3.1 Achieved Objectives

The objectives of this Thesis and how they were met are as follows

Assess the effect population sizes have on the performance and efficiency of GAs running within a fixed length of time with a slow fitness function. Chapter 3 assessed the effects that changing the population size and level of elitism has on GAs. It was discovered that lower population sizes perform better than larger population sizes when running within a fixed length of time but the lower population sizes have a lower level of efficiency.

Introduce new methods for improving the performance and efficiency of GAs with a slow fitness function. Chapters 4 and 5 introduced intelligent fitness functions and partial fitness functions.

Investigate the effects that the introduced methods have on GAs running within a fixed length of time. Chapters 4 and 5 investigated the effects that intelligent fitness functions and partial fitness functions have on GAs running within a fixed length of time.

Investigate the improvement provided by the methods to a GA with a slow fitness function. Chapter 6 investigated the improvements provided by intelligent fitness functions and partial fitness functions applied to a GA designed for breaking a monoalphabetic substitution cipher.

7.4 Future Work

Future work relating to intelligent fitness functions should include the following areas.

Hybrid GAs, as discussed earlier in this Chapter could benefit from the use of intelligent fitness functions. The possibility of the secondary method using an intelligent fitness function should be investigated. The advantages and disadvantages of the GA's intelligent fitness function and the secondary method's intelligent fitness function sharing the same memory should also be investigated.

Other Heuristics That Use a Fitness Function could benefit from the use of an intelligent fitness function. This is a large area to be researched as each heuristic would have to be investigated to decide if it would benefit from an intelligent fitness function.

Future work in the area of partial fitness functions should include the following areas.

Hybrid GAs can have the GA part improved by a partial fitness function. The fact that most hybrid GAs run the GA for a fixed length of time means that it is more likely to benefit from using a partial fitness function.

Other Heuristics may be able to use a partial fitness function if they use a population of solutions. Each heuristic would have to be investigated separately to decide if it could benefit from a partial fitness function.

Effects on unlimited GA runs will have to be investigated as the results in Chapter 6 indicated that as the number of full generations increase the benefits of a partial fitness function decrease. It would be interesting to see if there is a point where a standard GA surpasses the GA with a partial fitness function.

It would also be interesting to explore why sometimes the GA with a population size of 6 performs better than a population size of 60 in the same number of generations.

7.5 Conclusion

This Thesis has introduced intelligent fitness functions and partial fitness functions, both can improve the performance of GAs.

Intelligent fitness functions improve the performance of a GA by reducing the length of time the GA takes to complete a number of generations. The reduction is achieved by removing the need to re-evaluate the fitness of individuals seen before.

Partial fitness functions improve the performance of a GA by increasing the frequency that GAs are capable of taking evolutionary steps.

Bibliography

[1] Al-Kadi I.A. *The Origins of Cryptology: The Arab Contributions.* Cryptologia, vol 16, no. 2. pp. 97 – 126. 1992.

[2] Al-Kindī*A Manuscript on Deciphering Cryptographic Messages.*

[3] Antonisse J. *A New Interpretation of Schema Notation That Overturns the Binary Encoding Constraint.* Proceedings of the Third International Conference on Genetic Algorithms, Schaffer J.D. Ed. Morgan Kaufmann Publishers, pp. 86 – 91. 1989.

[4] Bagnall A.J. *The Applications of Genetic Algorithms in Cryptanalysis.* Master of Science Thesis. University of East Anglia. 1996.

[5] Baldwin J.M. "A New Factor in Evolution." American Naturalist 30. pp. 441 – 451. 1896.

[6] Belew R.K. and Mitchell M. (Eds.)*Adaptive Individuals in Evolving Populations.* Addison-Weslet. 1996.

[7] Biles J.A. *Genjam: A Genetic Algorithm for Generating Jazz Solos.* Proceedings of International Computer Music Conference. pp. 131 – 137. 1994.

[8] Birmingham J.A and Keny P. *Tree-searching and Tree-pruning Techniques.* Advances in Computer Chess. Clarke L.M. Ed. Edinburgh University Press, pp. 89 – 107. 1977.

[9] Branke J. *Creating Robust Solutions by Means of Evolutionary Algorithms.* Proceedings of parallel problem solving from nature. pp. 119 – 128. 1998.

[10] Branke J. Schmidt C. and Schmeck H. *Efficient Fitness Estimation in Noisy Environments.* Proceedings of Genetic and Evolutionary Computation. pp. 243 – 250. 2001.

[11] Bremermann H.J. *The Evolution of Intelligence. The Nervous System as a model of its environment.* Technical Report No.1, Contract No.477(17), Department of Mathematics, University of Washington, Seattle. 1958

[12] Bremermann H.J. *Optimisation Through Evolution and Recombination.* In, Self-Organising Systems, M.C. Yovits, G.T. Jacobi and G.D. Goldstiene, Eds. Washington, DC: Spartan Books, pp. 93 – 106. 1962.

[13] Bremermann H.J. *Quantitative aspects of goal-seeking self-organising systems.* Progress in Theoretical Biology, vol 1, New York: Academic Press, pp. 59 – 77. 1967.

[14] Bremermann H.J. *Numerical Optimisation Procedures Derived From Biological Evolution Processes.* Cybernetic Problems in Bionics, H.L. Oestereicher and D.R. Moore, Eds. New York: Gordon and Breach, pp. 543 – 562. 1968.

[15] Bremermann H.J. *On the Dynamics and Trajectories of Evolution Processes.* Biogenesis, Evolution, Homoeostasis. A. Locker, Ed. New York: Springer-Verlag, pp. 29 – 37. 1973.

[16] Bremermann H.J. and Rogson M. *An Evolution-Type Search Method for Convex Sets.* ONR Technical Report, Contracts 222(85) and 2656(58), UC Berkeley. 1964.

[17] Bremermann H.J. Rogson M. and Salaff S. *Search by Evolution.* Biophysics and Cybernetic Systems, M. Maxfield, A. Callahan and L.J. Fogel, Eds. Washington, DC: Spartan Books, pp. 157 – 167. 1965.

[18] Bremermann H.J. Rogson M. and Salaff S. *Global Properties of Evolution Processes.* Natural Automata and Useful Simulations, H.H. Pattee, E.A. Adlsack, L. Fein and A.B. Callaham, Eds. Washington, DC: Spartan Books, pp. 3 – 41. 1965.

[19] Broyden C.G. *Journal of the Institute for Mathematics and Applications.* Volume 6, pp. 222 – 231. 1970.

[20] Bull L. *On Modal-Based Evolutionary Computation.* Soft computing, vol 3. pp. 76 – 82. 1999.

[21] Brooks J. L. *Just Before the Origin: Alfred Russel Wallace's Theory of Evolution.* Columbia University Press, 1984.

[22] Cantu-Paz E. *On Random Numbers and the Performance of Genetic Algorithms.* Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 311 – 318. 2002.

[23] Cantu-Paz E. Goldberg D.E. and Harik G. *The Gamblers Ruin Problem, Genetic Algorithms, and the Sizing of Populations* Proceedings of the 1997 IEEE International Conference on Evolutionary Computation. pp. 7 – 12. 1997.

[24] Caprara A., Fischetti M., and Maio D. *Exact and approximate algorithms for the index selection problem in physical database design.* IEEE Transactions on Knowledge and Data Engineering, 7(6). 1995.

[25] Carter F. and Gallehawk J. *The Enigma Machine and the Bombe.* The Bletchley Park Trust Reports. Report No. 9. 1998.

[26] Colorni A. Dorigo M. and Maniezzo V. *Distributed Optimisation by Ant Colonies* Proceedings of the First European Conference on Artificial Life. Varela F. and Bourgine P. Eds. Elsevier Publishing, pp. 134 – 142. 1992.

[27] Commer D.E. *Computer Networks and Internets: with Internet Applications* Upper Saddle River, N.J. ; London. ISBN 01312367X. 2001

[28] Darwin C. *The Origin Of Species.* Oxford University Press, Walton Street, Oxford. OX2 6DP, UK. Based on: On The Origin Of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. Second Edition, London 1859. First Published 24 Nov 1859.

[29] Davis L. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold. 1991.

[30] Dawkins R. *Climbing Mount Improbable.* The Penguin Group, 1996. ISBN 0-670-85018-7.

[31] Dawkins R. *The Blind Watchmaker.* Longman Scientific and Technical, 1986. ISBN 0-582-44694-5.

[32] Dawkins R. *The Extended Phenotype.* Oxford Press, 1989. ISBN 0-19-286088-7.

[33] Dawkins R. *The Selfish Gene.* Oxford Press, 1976. ISBN 0-330-36710-2.

[34] De Jong K.A. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Doctoral Dissertation, University of Michigan. Dissertation Abstracts International 36(10), 5140b. 1975.

[35] Dennis J. and Torczon V. *Managing Approximate Models in Optimisation.* Multidisciplinary design optimisation: State-of-the-Art. pp. 330 – 347. 1997.

[36] *Data Encryption Standard (DES).* Federal Information Processing Standards Publication 46-2. 1993.

[37] Fitzpatrick J.M. Grefenstette J.J. *Genetic Algorithms in Noisy Environments.* Machine Learning, vol 3. pp. 101 – 120. 1988.

[38] Fletcher R. *Practical Methods of Optimization.* Chirchester, UK: John Wiley and Sons. 1987.

[39] Fleurent C. and Ferland J. *Genetic Hybrids for the Quadratic Assignment Problem.* DIMACS Series in Mathematics and Theoretical Computer Science. Volume 16. pp. 190 – 206. 1994.

[40] Fogel D. (Ed) *The Fossil Record.* IEEE Press, 1998.

[41] Fogel L.J. Owens A.J. and Walsh M.J. *Artificial Intelligence Through Simulated Evolution* John Wiley & Sons, Inc. 1966.

[42] Forman S.L. *Congressional Redistricting Using a TSP-based Genetic Algorithm.* Proceedings of the Genetic and Evolutionary Computations Conference. pp. 1262. 2002.

[43] Fraser A.S. *Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction.* Australian Journal of Biological Science. Vol 10, pp. 484 – 491. 1957.

[44] Fraser A.S. *Simulation of Genetic Systems by Automatic Digital Computers. II. Effects of Linkage on Rates of Advanced Under-selection.* Australian Journal of Biological Science. Vol 10, pp. 492 – 499. , 1957.

[45] Fraser A.S. *Simulation of Genetic Systems by Automatic Digital Computers. IV. Epistasis.* Australian Journal of Biological Science. Vol 13, pp. 329 – 346. 1960.

[46] Fraser A.S. *Simulations of Genetic Systems.* Journal of Theoretical. Biological Science. Vol2 pp. 329 – 346. 1962.

[47] Fraser A.S. *The Evolution of Purposive Behaviour.* In, Purposive Systems, H.von Forester, J.D. White, L.J. Peterson and J.K.Russell, Eds., Washington, DC: Spartan Books, pp. 15-23. 1968.

[48] Glover F. *Tabu Search - Part I.* ORSA Journal on Computing, 1(3). pp. 190 – 206. 1989.

[49] Glover F. *TABU Search - Part II.* ORSA Journal on Computing 2(1). pp. 4 – 32. 1990.

[50] Glover F., Kelly J. and Laguna M. *Genetic Algorithms and Tabu Search: Hybrids for Optimisation.* Computers Ops Res. 22. 1995.

[51] Goldberg D.E. Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley, 1989. ISBN 0-201-15767-5.

[52] Goldberg D.E. *The Design of Innovation: Lessons from Genetic Algorithms, Lessons for the Real World.* IlliGAL Report No. 98004, 1998.

[53] Goldberg, D.E. *Using Time Efficiently: Genetic-Evolutionary algorithms and the continuation problem.* GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufman. Pages 212–219, (1999). `http://citeseer.nj.nec.com/article/goldberg99using.html`

[54] Goldberg, D.E. Deb, K. and Clark, J. H. *Genetic Algorithms, Noise, and the sizing of populations.* Complex Systems, 6, Pages 333–362. (1992)

[55] Gomez A. and De La Fuente D. *Resolution of Strip-Packing Problems With Genetic Algorithms.* Journal of the Operational Research Society, vol 51. pp. 1289 – 1295. 2000.

[56] Grefenstette J.J. and Fitzpatrick J.M. *Genetic Search with Approximate Function Evaluations.* 1st International Conference On Genetic Algorithms and Their Applications, Pittsburgh. pp. 112 – 120. 1985.

[57] Grierson D.E. and Pak W.H. *Optimal Sizing Geometrical and Topological Design Using a Genetic Algorithm.* Structural Optimisation , vol 6, no. 3. pp. 151 – 159. 1993.

[58] Grūndlingh W.R. and Van Vuuren J.H.*Using Genetic Algorithms to Break a Simple Cryptographic Cipher.* `http://dip.sun.ac.za/~vuuren/papers/genetic.ps`

[59] Guha A., Harp S.A. and Samad T. *Genetic Synthesis of Neural Networks.* Technical Report, Honeywell Corporate Systems Development Division. 1988.

[60] Hajela P. and Lee J. *Genetic Algorithms in Multidisciplinary Rotor Blade Design.* Proceedings of 36^{th} Structures, Structural Dynamics and material Conference. New Orleans. 1998.

[61] Harp S.A., Samad T. and Guha A. *Towards the Genetic Synthesis of Neural Networks.* Proceedings of the 3^{rd} International Conference on Genetic Algorithms pp. 360 – 369 1989.

[62] Hart W.E. and Belew R.K. *Optimisation with Genetic Algorithm Hybrids Using Local Search.* Adaptive Individuals in Evolving Populations, Belew R.K. and Mitchell M. (eds.) pp. 483 – 494. 1996.

[63] Hassan K. and Conner M. *Predicting Feedback Polynomial of LFSR Using Genetic Algorithm.* Approximation and Learning in Evolutionary Computation Workshop, GECCO 2002, Barry A. Ed. pp. 9 – 11. 2002.

[64] Hillermeier C. and Keppler J. *An Application of Genetic Algorithms and Neural Networks to Scheduling Power Generating Systems.* Parallel Problem Solving From Nature. pp. 811 – 818. 1996.

[65] Hinton G.E. and Nowlan S.J. *How Learning Can Guide Evolution.* Computer Systems, 1(1) pp. 495 – 502. 1987.

[66] Ho A.C.H. and Kwong S. *Optimisation of CDMA Based Wireless Systems.* Proceedings of the Genetic and Evolutionary Computations Conference. pp. 1266. 2002.

[67] Holland J. H. *Adaption in Natural and Artificial Systems.* A Bradford Book, The MIT Press. ISBN 0-262-08213-6.

[68] Ibaraki T. *Combinations With Other Optimization Methods.* Handbook of Evolutionary Computations. Back T., Fogel D.B. and Michalewicz Z. eds. Bristol and New York: Institute of Physics Publishing and Oxford University Press. pp. D:31 – D:32. 1997.

[69] Jin Y. Olhofer M. and Sendoff B. *On Evolutionary Optimisation With Approximate Fitness Functions.* Proceedings of the Genetic and Evolutionary Computation Conference. pp. 786 – 792. 2000.

[70] Jin Y. Olhofer M. and Sendhoff B. *Managing Approximate Models in Evolutionary Aerodynamical Design Optimisation.* Proceedings of IEEE Congress on Evolutionary Computation, vol 1. pp. 592 – 599. 2001.

[71] Jin Y. *Fitness Approximation in Evolutionary Computation – A Survey* Approximation and Learning in Evolutionary Computation Workshop, GECCO 2002, Barry A. Ed. pp. 3 – 4. 2002.

[72] Jin Y. and Sendhoff B. *Fitness Approximation in Evolutionary Computing – A Survey.* Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 1105 – 1112. 2002.

[73] Johanson B. and Poli R. *GP Music: An Interactive Genetic Programming System for Music Generation With Automated Fitness Raters.* Proceedings of the 3rd Annual Conference on Genetic Programming. pp. 181 – 186. 1998.

[74] Kahn D. *The Codebreakers.* Macmillan, ISBN 0-025-60460-0. 1996.

[75] Kaniel S. *Estimates for Some Techniques in Linear Algebra.* Math Comput. 20 pp. 369 – 378. 1966.

[76] Kennedy, J. and Eberhart R.C. *Genetic Programming Using Genotype - Phenotype Mapping from Linear Genomes into Lenear Phenotypes.* Genetic Programming. 1996.

[77] Kirkspatrick S. Gelatt Jr. C. D. and Vecchi M.P. *Optimisation by Simulated Annealing.* Science, 220, 4598, pp. 671 – 680. 1983.

[78] Knodler K. Poland J. and Zell A. *Memetic Algorithms for Combinatorial Optimisation Problems in the Calibration of Modern Combustion Engines.* Proceedings of

the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 687. 2002.

[79] Kodiyalam S. Nagendra S. and DeStefano j. *Composite Sandwich Structural Optimisation with Application to Satellite Components.* AIAA Journal, vol 34, no. 3. pp. 614 – 621. 1996.

[80] Koza J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press. 1992.

[81] Koza J.R. *Genetic Programming II : Automatic Discovery of Reusable Programs* MIT Press. 1994.

[82] Krarup J. and Pruzan P.M. *The Simple Plant Location Problem: Survey and Synthesis.* European Journal of Operation Resource 12. pp. 36 – 81. 1983.

[83] Kratica J. *Improving Performances of the Genetic Algorithm by Caching.* Computers and Artificial Intelligence Vol 18, no. 3. pp. 271 – 283. 1999.

[84] Kratica J., Tosic D., Filipovic V. and Ljubic I. *Solving the Simple Plant Location Problem by Genetic Algorithm.* RAIRO Operations Research, 35. pp. 127 – 142. 2001.

[85] Kratica J., Ljubic I. and Tosic D. *A Genetic Algorithm for the Index Selection Problem.* `http://citeseer.ist.psu.edu/568873.html`

[86] Lamarck, J. B. Zoological Philosophy. London. 1809.

[87] Land M. *Evolutionary Algorithms with Local Search for Combinatorial Optimisation.* Doctoral Dissertation, University of California, San Diego, CA. 1998.

[88] Langdon W.B. and Qureshi A. *Genetic Programming - Computers Using "Natural Selection" to Generate Programs.* Research Note : RN/95/76. Department of Computer Science, University College London. 1995.

[89] Lee J. and Hajela P. *Parallel Genetic Algorithms Implementation for Multidisciplinary Rotor Blade Design.* Journal of Aircraft, vol 33, no. 5. pp. 962 – 969. 1996.

[90] Liang K.H. Yao X. and Newton C. *Evolutionary Search of Approximated n-dimensional Landscape.* International Journal of Knowledge based Intelligent Engineering Systems, vol 4, no. 4. pp. 172 – 183. 2000.

[91] Linden D.S. *Antenna Design Using Genetic Algorithms.* Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 1133 – 1140. 2002.

[92] Ljubic I., Raidl G.R. and Kratica J. *A Hybrid* GA *for the Edge-Biconnectivity Augmentation Problem.* Parallel Problem Solving from Nature. pp. 641 – 650. 2000.

[93] Louis S.J.. *Genetic Learning for Combinational Logic Design* Approximation and Learning in Evolutionary Computation Workshop, GECCO 2002, Barry A. Ed. pp. 21 – 26. 2002.

[94] Louis S.J., McGraw G. and Wyckoff R.O. *Case-based Reasoning Assisted Explanation of Genetic Algorithm Results.* Technical Report No. 361. Bloomington, Indiana University. 1992.

[95] Malthus T. *Essay on the Principle of Population*, 1798.

[96] Mathias K.E., Whitley D.L., Stork C. and Kusuma. T. *Staged Hybrid Genetic Search for Seismic Data Imaging.* Proceedings of the 1994 IEEE International Conference on Evolutionary Computation. 1994.

[97] Maxwell III S.R. *Experiments with a coroutine model for genetic programming.* In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, IEEE Press. Volume 1, pp. 413 – 417. 1994.

[98] Mendel G. *Experiments in plant hybridisation.* Oliver and Boyd, 1965.

[99] Menzel D. (ed) *Fundamental Formulas of Physics.* Volume 2, 2^{nd} edition. 1960.

[100] Metropolis N., Rosenbluth A.W. Teller M.N. and Teller W. *Equations of State Calculations by Fast Computing Machines.* Journal of Chemical Physics volume 21. pp. 1087 – 1091. 1953.

[101] Meysenburg M.M. Hoelting D. McElvain D. Foster J.A. *How Random Generator Quality Impacts Genetic Algorithm Performance.* Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 480 – 483. 2002.

[102] Meysenburg M.M. Hoelting D. McElvain D. Foster J.A. *A Genetic Algorithm-Specific Test of Random Number Generator Quality.* Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 691. 2002.

[103] Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag. 1992.

[104] Mogul J. and Deering S. *RFC 1191 - Path MTU Discovery* `http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1191.html`

[105] Montana D.J. *Automated Parameter Tuning for Interpretation of Synthetic Images.* Handbook of Genetic Algorithms. Davis L. Ed. Van Nostrand Reinhold, pp. 282 – 311. 1991

[106] Montana D. and Davis L. *Training Feed Forward Neural Networks Using Genetic Algorithms.* Proceedings of the 11th International Joint Conference on Artificial Intelligence. pp. 762 – 767. 1989.

[107] Moscato P. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.* Technical Report 790. Caltech Concurrent Computation Program. 1989.

[108] Nair P.B. and Keane A.J. *Combining Approximation Concepts with Algorithm-based Structural Optimisation Procedures.* Proceedings of 39th AIAA/ASMEASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference. pp. 1741 – 1751. 1998.

[109] Newton I. *Methodus Fluxionum et Serierum Infinitarum.* 1664-1671.

[110] Newton M., Sykora O., Withall M. and Imrich V. *A Parallel Approach to Row-Based VLSI Layout Using Stochastic Hill-Climbing.* IEA2003AIE. 2003.

[111] Ochoa G. *Setting the Mutation Rate: Scope and Limitations of the $\frac{1}{l}$ Heuristic* Proceeding of the Genetic and Evolutionary Computation Conference. pp. 495 – 502. 2002.

[112] Orvosh D. and Davis L. *Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints* Proceedings of the Fifth International Conference on Genetic Algorithms pp. 650 . 1993.

[113] Pierret S. *Turbomachinery Blade Design Using a Navier-Stokes Solver and Artificial Neural Network.* ASME Journal of Turbomachinery, vol 121, no. 3. pp. 326 – 332. 1999.

[114] Piras R.A. *A Comparison of Genetic Algorithms and Neural Networks as used in Optimisation* Final Year Project Report, Department of Computer Science, Loughborough University. 1993.

[115] Porto V.W. *Neural-Evolutionary Systems: New Areas for Evolutionary Computation Research in Neural Systems.* Handbook of Evolutionary Computation, Back T., Fogel D.B. and Michalewicz Z eds. Bristol and New York: Institute of Physics Publishing and Oxford University Press. pp. D1.3:1 – D1.3:2. 1997.

[116] Powell M.J.D. *An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives* The Computer Journal. 7 pp. 155 – 162. 1964.

[117] *IP Quality of Service - FAQ* `http://www.inf.ufrgs.br/granvile/QoS/Imprimir/faq.htm`

[118] Radcliffe N.J. and Surry P.D. *Formal Memetic Algorithms.* Evolutionary Computing: AISB Workshop. pp. 1 – 16. 1994.

[119] Ramsey C.L. and Grefenstette J.J. *Case-Based Initialisation of Genetic Algorithms.* Proceedings of the 5^{th} International Conference on Genetic Algorithms. pp. 84 – 91. 1993.

[120] Ratle A. *Accelerating the Convergence of Evolutionary Algorithms by Fitness Landscape Approximation.* Parallel Problem Solving from Nature. vol 5. pp. 87 – 96. 1998.

[121] Ratle A. *Optimal Sampling Strategies for Learning a Fitness Model.* Proceedings of 1999 Congress on Evolutionary Computation, vol 3. pp. 2078 – 2085. 1999.

[122] Rasheed K. Ni X. and Vattam S. *Comparison of Methods for using Reduced Models to Speed Up Design Optimisation.* Approximation and Learning in Evolutionary Computation Workshop, GECCO 2002, Barry A. Ed. pp. 17 – 20. 2002.

[123] Rechenburg I. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Diologischen Evolution.* Fromman Holzboog. 1973.

[124] Redmond J. and Parker G. *Actuator Placement Based on Reachable Set Optimisation for Expected Disturbance.* Journal Optimisation Theory and Applications, vol 90, no. 2. pp. 279 – 300. 1996.

[125] Reed J. Toombs R. and Barricelli N.A. *Simulation of Biological Evolution and Machine Learning.* Journal of Theoretical Biology, Vol. 17, pp. 319 – 342. 1967.

[126] Reeves C.R. *Using Genetic Algorithms with Small Populations.* Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan-Kaufman. Pages 92–99, (1993).

[127] Reeves C.R. and Taylor S.J. *Selection of Training Data for Neural Networks by a Genetic Algorithm.* Parallel Problem Solving From Nature. pp. 633 – 642. 1998.

[128] Renders J. and Bersini H. *Hybridizing Genetic Algorithms with Hill Climbing Methods for Global Optimisation: Two Possible Ways.* Proceedings of the 1st IEEE Conference on Evolutionary Computation. pp. 312 – 317. 1994.

[129] Rogers D. *GSPLINES: A Hybrid of Friedman's Multivariate Adaptive Regression Splines (MARS) Algorithm with Holland's Genetic Algorithm.* Proceedings of the 4th International Conference on Genetic Algorithms. pp. 384 – 391. 1991.

[130] Rosenblatt F. *The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain.* Psychological Review, 65 pp. 386 – 407. 1958.

[131] *RSA Cryptography Standard.* PKCS #1 v2.1 RSA Laboratories. 2002.

[132] Rudnick M. *A Bibliography of the intersection of Genetic Search and Artificial Neural Networks.* Technical Report No. CS/E 90-001. Beaverton: Oregon Graduate Center, Department of Computer Science and Engineering. 1990.

[133] Sano Y. and Kita H. *Optimisation of Noisy Fitness Functions by Means of Genetic Algorithms Using History.* Parallel Problem Solving from Nature. vol 1917 of Lecture Notes in Computer Science, Springer. 2000.

[134] Shyy W. Tucker P.K. Vaidyanathan R. *Response Surface and Neural Networks Techniques for Rocket Engine Injector Optimisation.* Technical Report 99-2455, AIAA. 1998.

[135] Simpson R. and Yachavaram S. *Faster Shellsort Sequences: A Genetic Algorithm Application.* Proceedings of the International Society for Computers and their Applications (ISCA). 1999.

[136] Singh S. *The Code Book.* Fourth Estate Limited. ISBN 1-85702-879-1. 1999.

[137] Sinha A. and Goldberg D.E. *A Survey of Hybrid Genetic and Evolutionary Algorithms.* IlliGAL Report No. 2003004. Illinois Genetic Algorithms Laboratory, University of Illinois. 2003.

[138] Sirag D.J and Weisser P.T. *Towards a Unified Thermodynamic Genetic Operator.* Proceedings of the 2^{nd} International Conference on Genetic Algorithms. pp. 116 – 122. 1987.

[139] Souza P.S.D. and Talukadar S.N. *Genetic Algorithms in Asynchronous Teams.* Proceedings of the 4th International Conference on Genetic Algorithms. pp. 392 – 397. 1991.

[140] Syswerda G. *Schedule Optimisation Using Genetic Algorithms.* Handbook of Genetic Algorithms. Davis L. Ed. Van Nostrand Reinhold, pp. 332 – 349. 1991.

[141] Thangiah S.R. *Vehicle Routing with Time Windows Using Genetic Algorithms.* Application and Book of Genetic Algorithms, vol 2, ed. Chambers L. pp. 253 – 277. 1995.

[142] Takagi H. *Interactive Evolutionary Computation.* Proceedings of the 5th International Conference on Soft Computing and Information / Intelligent Systems. pp. 41 – 50. 1998.

[143] Tanenbaum A. *Modern Operating Systems.* Prentice-Hall, NJ. 1992.

[144] Turing A.M. *Intelligent Machinery.* Collected Works of A.M. Turing: Mechanical Intelligence, Ince D.C. Ed. pp. 107 – 127. 1992.

[145] Vignaux G.A. and Michalewicz Z. *A Genetic Algorithm for the Linear Transportation Problem.* IEEE Trans. on Systems, Man and cybernetics, Vol. 21, no 2, pp. 445 – 452. 1991.

[146] Whitley D., Gordon V.S., and Mathias K. *Lamarckian evolution, the Baldwin Effect and Function Optimization.* Parallel Problem Solving from Nature - 94. pp. 6 – 15. 1994.

[147] Withall M.S. *The Evolution of Complete Software Systems.* PhD Thesis. Department of Computer Science, Loughborough University. 2003.

[148] Wright A.H. *Genetic Algorithms for Real Parameter Optimisation.* Foundations of Genetic Algorithms. Rawlins G.J.E. Ed. Morgan Kaufmann Publishers, pp. 205 –218. 1991.

[149] Yagiura M. and Ibaraki T. *Use of Dynamic Programming in Genetic Algorithms for Permutation Problems.* European Journal of Operational Research. 92(2) pp. 387 – 401. 1996.

[150] Yao X. *Evolving Artificial Neural Networks.* Proceedings of the IEEE 87(9). pp. 1423 – 1447. 1999.

[151] Yen J., Liao J., Randolph D. and Lee B. *A Hybrid Approach to Modelling Metabolic Systems Using Genetic Algorithms and Simplex Method.* Proceedings of the 11th IEE Conference on Artificial Intelligence for Applications pp. 277 – 283. 1995.

[152] Yilmaz A.S. and Wu A.S. *A Comparison of Haploidy and Diploidy without Dominance on Integer Representations.* `http://citeseer.ist.psu.edu/581298.html`.

Appendix A

Publications

- **Comparison of evolving against peers and fixed opponents using Corewars.**

  J.L. Cooper and C.J. Hinde. Genetic and Evolutionary Computation Conference (GECCO) 2002

- **Improving the Efficiency of Genetic Algorithms using Intelligent Fitness Functions.**

  J.L Cooper and C.J. Hinde. The 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE) 2003.

- **Packet Transmission Optimisation Using Genetic Algorithms.**

  M.S. Withall, C.J. Hinde, R.G. Stone and J.L. Cooper. The 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE) 2003.

- **Investigation into the effects of varying the parameters of packets travelling across the Internet.**

  J.L. Cooper, M.S. Withall, C.J. Hinde, R.G. Stone. Internal Report num:1070, Computer Science Department, Loughborough University.

Appendix B

Full Results for Chapter 3

B.1 Population Sizes and their Effects on Performance

This section shows and discusses the results of the experiments examining the effects of population sizes on the performance of the GA.

B.1.1 One Max Problem

The following graphs show the results of the runs of the GA, with the varying population sizes, against the One Max Problem.

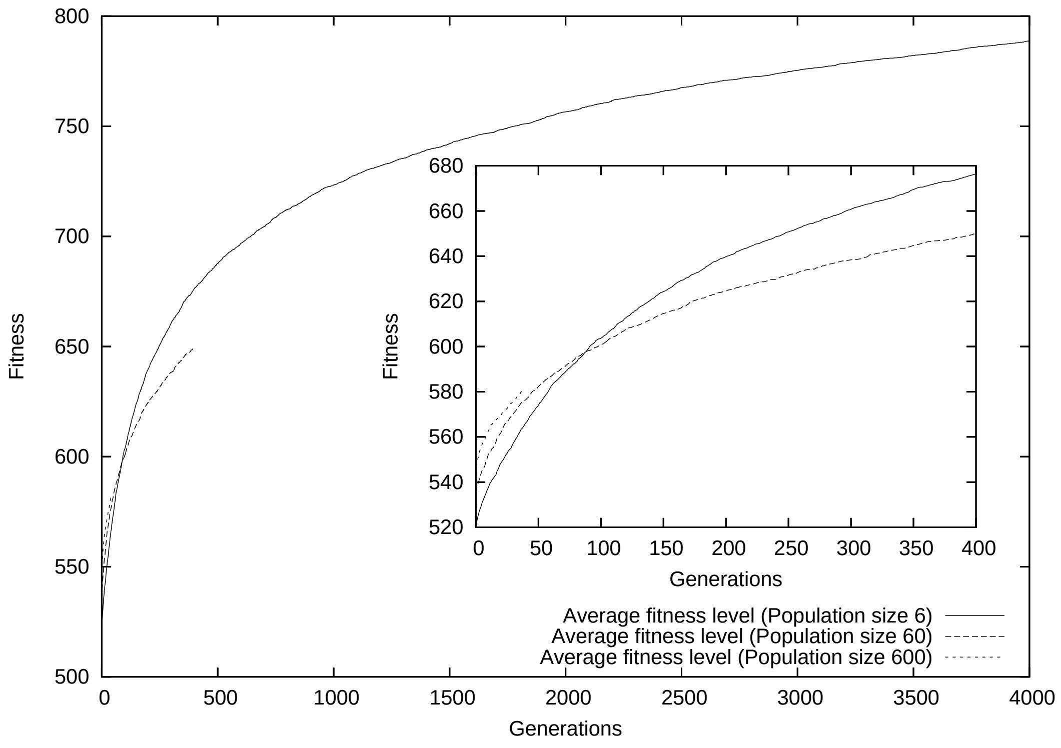

Figure B.1: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against generations

The graph in Figure B.1 shows the average best fitness of all three population sizes. They are shown against generations to show the difference in the number of generations the different population sizes can evolve for in the same length of time. While the higher population sizes do marginally better to start with, the population size of 6 manages to improve its fitness to a higher level than the other population sizes with the extra generations it can achieve in the time available to it. It is interesting to note that the population size of 6 performs better than the population size of 60 within the same number of generations, this is discussed later.

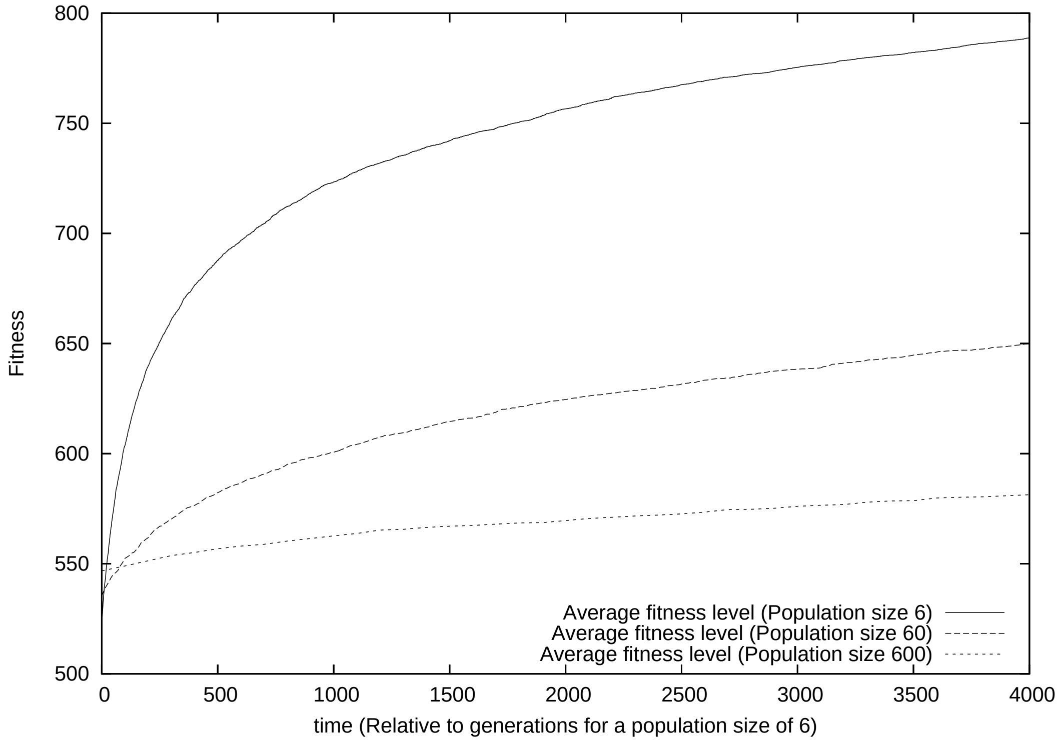

Figure B.2: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph in Figure B.2 shows the average best fitness of all three population sizes. They are shown against the time taken to run the GA. This graph shows that the population size of 6 does better than the higher population sizes when compared on time taken.

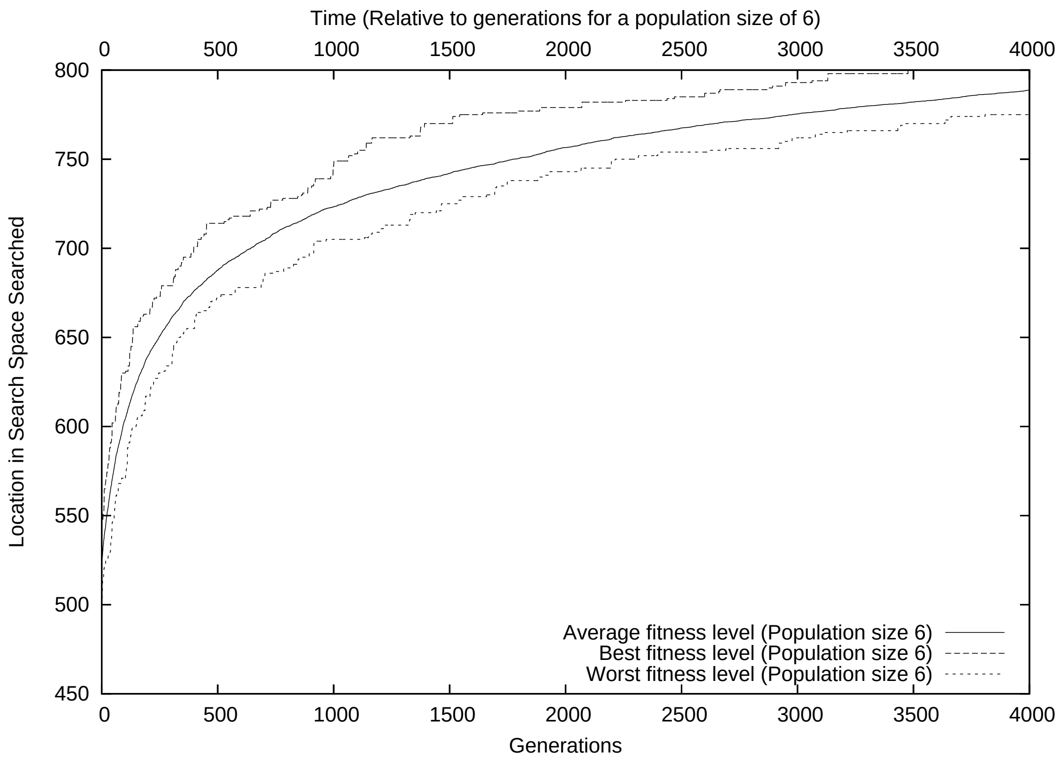

Figure B.3: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 6

The graph in Figure B.3 shows the average best fitness level of the GA with a population size of 6, the best fitness level the GA achieved with a population size of 6 and the worst best fitness level the GA achieved with a population size of 6.

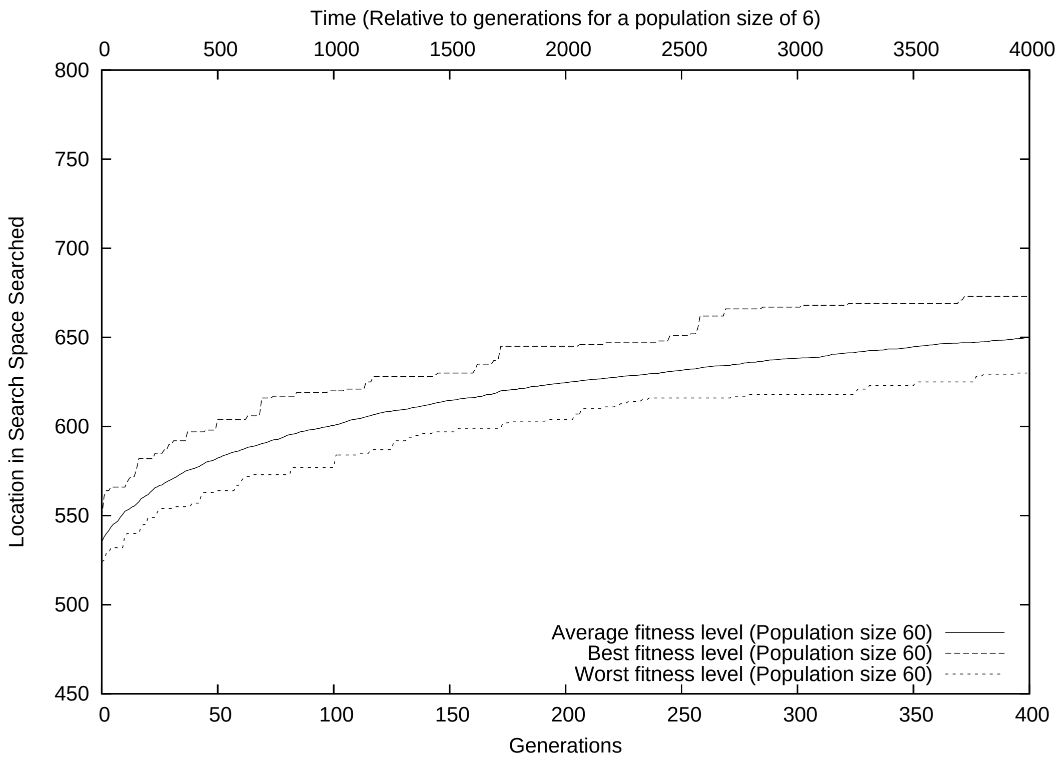

Figure B.4: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 60

The graph in Figure B.4 shows the average best fitness level of the GA with a population size of 60, the best fitness level the GA achieved with a population size of 60 and the worst best fitness level of the 50 runs the GA achieved with a population size of 60.

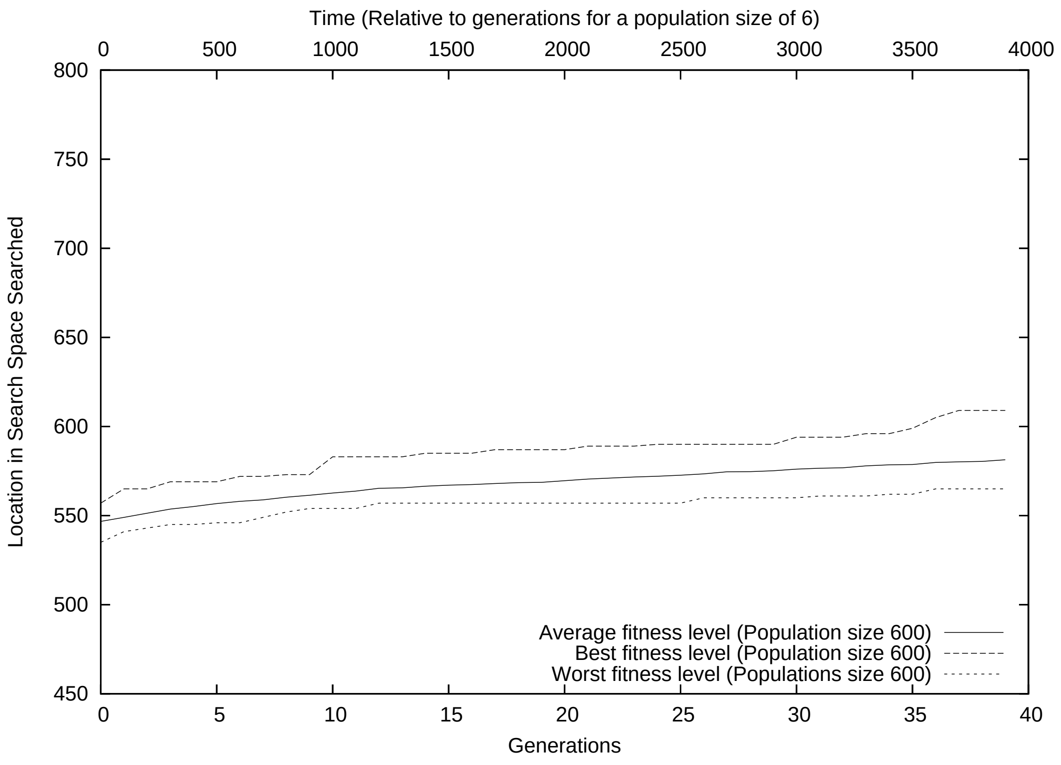

Figure B.5: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 600

The graph in Figure B.5 shows the average fitness level of the GA with a population size of 600, the best fitness level the GA achieved with a population size of 600 and the worst best fitness level of the 50 runs the GA achieved with a population size of 600.

Population Size	6	60	600
40 Generations	557	570	576
400 Generations	676	649	—
4000 Generations	788	—	—

Table B.1: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table B.1 shows the average fitness levels of the three different population sizes at forty, four hundred and four thousand generations. While at 40 Generations the best results is that of the GA with a population size of 600 followed by that of the GA with a population size of 60 and the population size 6 performing the worst.

The GA with a population size of 600 has used up all the time available to it by the 40^{th} generation and as such has not been able to improve on its fitness level of 576. The GA with a population size of 60 has reached a fitness level of 649 but the GA with a population size of 6 has produced a fitness level of 676. This is interesting as the lower population size has achieved a higher fitness level than the population size of 60 in the same number of generations, which will have taken it $\frac{1}{10}$ of the time to run. This result is counter intuitive and will need to be researched more to understand why it occurs.

The GA with a population size of 60 has used up all the time available to it by the 400^{th} generation and as such has not been able to improve on its fitness level. At its 4000^{th} generation the GA with a population size of 6 has reached the highest fitness achieved by the runs.

B.1.2 Deceptive Trap Functions

The following graphs show the results of the runs of the GA, with the varying population sizes, with the Deceptive Trap Function as a fitness function.

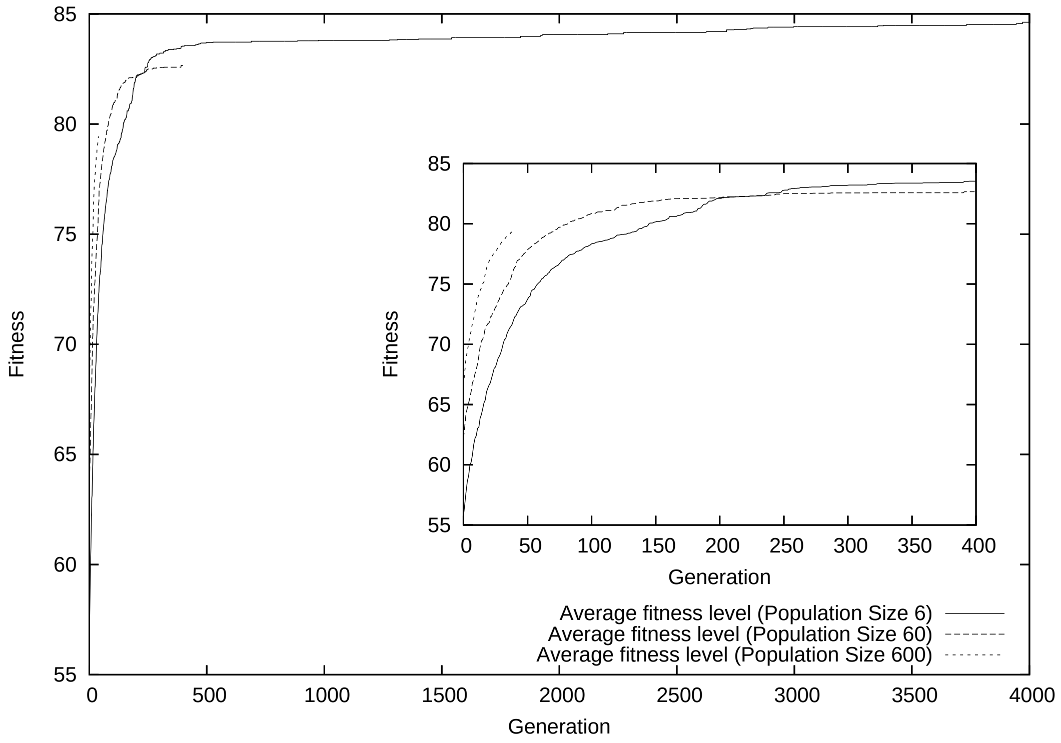

Figure B.6: Average Fitness levels, of the best of each of 50 runs, for all three population sizes, shown against generations

The graph in Figure B.6 shows the average fitness of all three population sizes. They are shown against generations to show the difference in the number of generations the different population sizes can evolve for in the same length of time. Just like the GA with the one max problem the higher population sizes do better at the start, but the population size of 6 manages to improve its fitness to a higher level than the other population sizes with the extra generations it can achieve in the time available to it. Again the population size of 6 surpassed the performance of the population size of 60 within the same number of generations. This is discussed later.

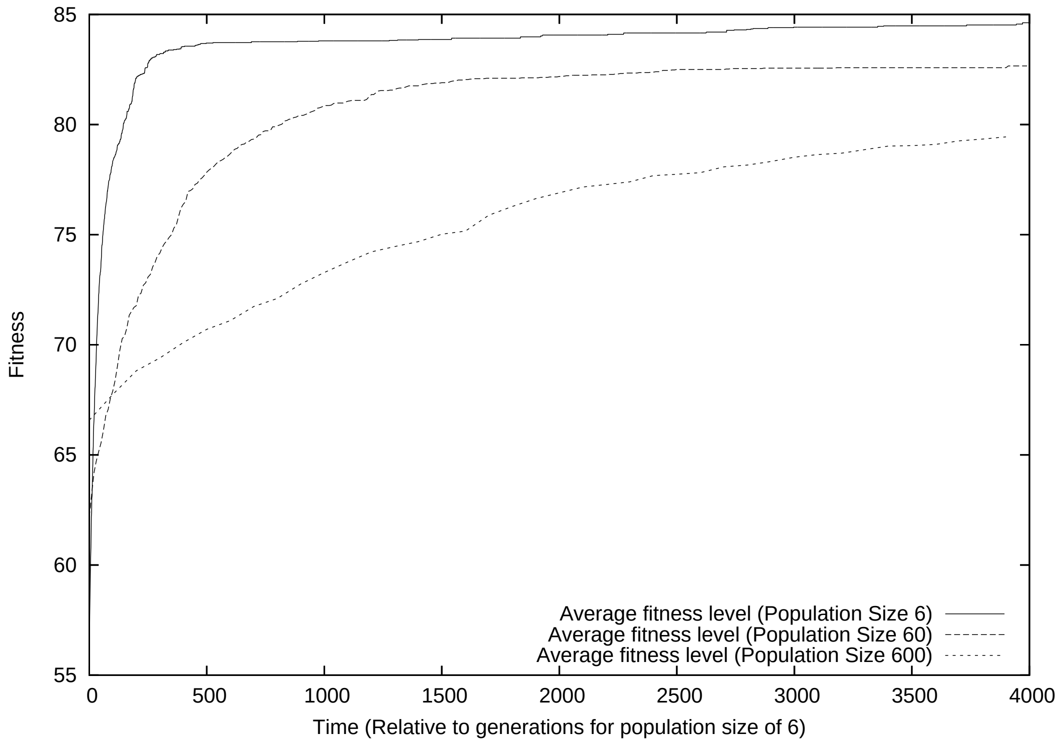

Figure B.7: Average Fitness levels, of the best of each of 50 runs, for all three population sizes, shown against time

The graph in Figure B.7 shows the average fitness of all three population sizes against the time taken to run the GA. Again the population size of 6 performs better than the higher population sizes in a set length of time.

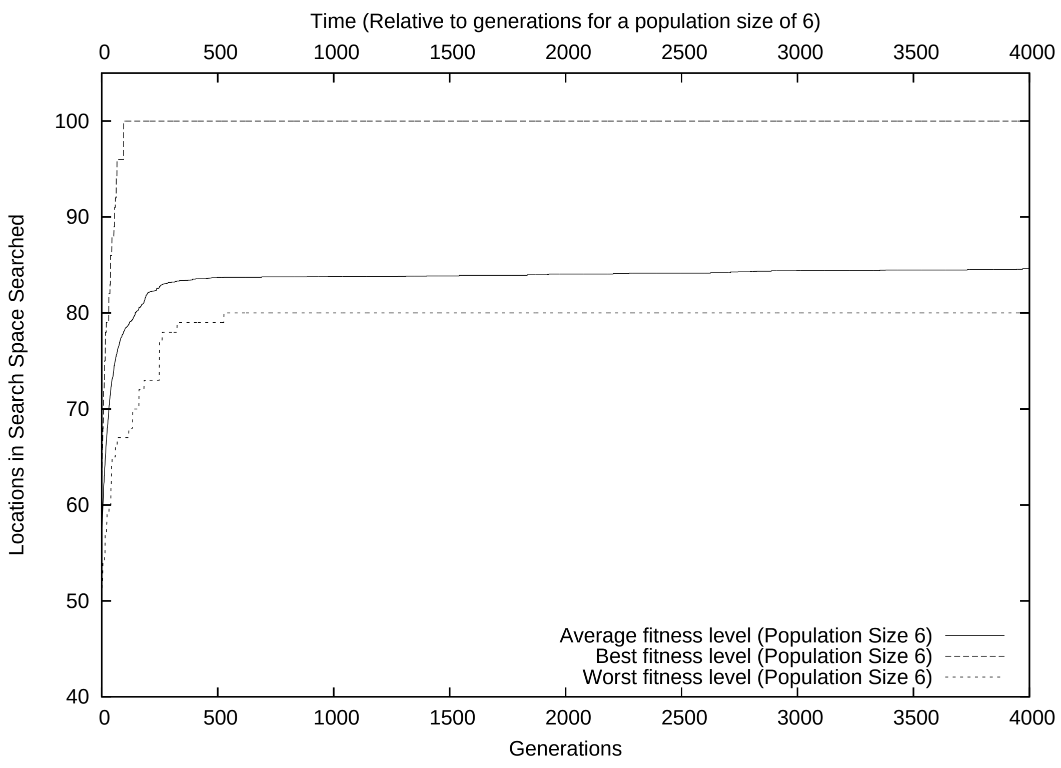

Figure B.8: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 6

The graph in Figure B.8 shows the average fitness level of the GA with a population size of 6, the best fitness level the GA achieved with a population size of 6 and the worst best fitness level of the 50 runs the GA achieved with a population size of 6.

The levelling off of the average fitness level and the worst fitness level is due to the GA getting trapped at a local minimum and failing to escape. The levelling off for the best fitness level is due to the GA solving the problem and as such has reached the highest fitness level.

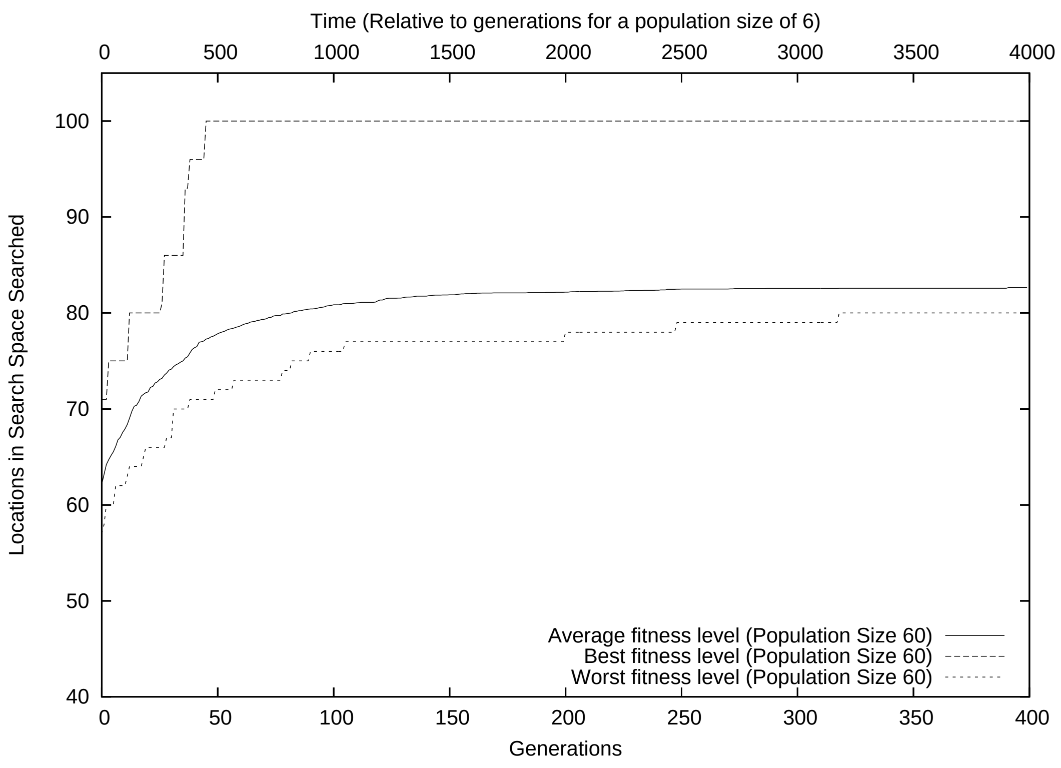

Figure B.9: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 60

The graph in Figure B.9 shows the average fitness level of the GA with a population size of 60, the best fitness level the GA achieved with a population size of 60 and the worst best fitness level of the 50 runs the GA achieved with a population size of 60.

Again the levelling off of the average fitness level is due to the GA getting stuck at local minimum. While the levelling off of the best fitness level is due to it solving the problem.

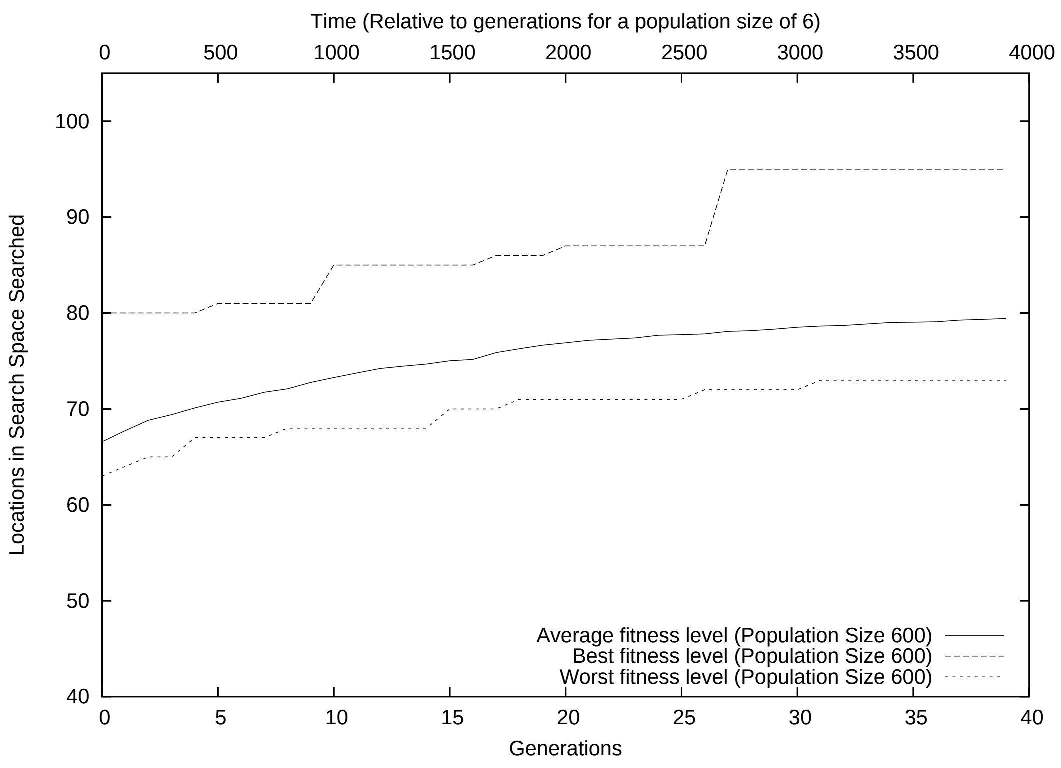

Figure B.10: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 600

The graph in Figure B.10 shows the average fitness level of the GA with a population size of 600, the best fitness level the GA achieved with a population size of 600 and the worst best fitness level the GA achieved with a population size of 600. It is interesting to note that the GA with a population size of 600 fails to achieve the maximum score with any of its runs despite having a higher initial best fitness level.

Population Size	6	60	600
40 Generations	72	76	79
400 Generations	83	82	—
4000 Generations	84	—	—

Table B.2: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table B.2 shows that after all three population sizes had evolved for 40 generations the population size of 600 had achieved the highest fitness level.

The GA with a population size of 600 has used up all of the time available to it by 40 generations and never reaches 400 generations. So it has not been able to improve upon its fitness level of 79. Both the GAs using a population size of 6 and 60 have overtaken it at this stage. At the 400 generation stage the GA with a population size of 6 has also overtaken the GA with a population size of 60. It is important to remember that the GA with a population size of 6 has taken $\frac{1}{10}$ of the time that the GA with a population size of 60 has taken to get to this stage.

The GA with a population size of 60 has used up all the time available to it by 400 generations and never reaches 4000 generations. So it has not been able to improve upon its fitness level. The GA with a population size of 6 has been able to improve slightly again, reaching the fitness level of 84, which is the highest encountered.

The best run for the GA with a population size of 6 reached the maximum possible fitness level in the 95^{th} generation. The best run for the GA with a population size of 60 reached the maximum possible fitness level in the 45^{th} generation. While this is a difference of 50 generations, the best run of the GA with a population size of 6 reached the maximum fitness level in 3% of the time available to it while the best run of the GA with a population size of 60 reached the maximum possible fitness level in 12% of the available time.

B.1.3 GA Hard Problem

The following graphs show the results of the runs of the GA, with the varying population sizes, against the GA Hard Problem. The default parameters were used for the GA Hard Problem.

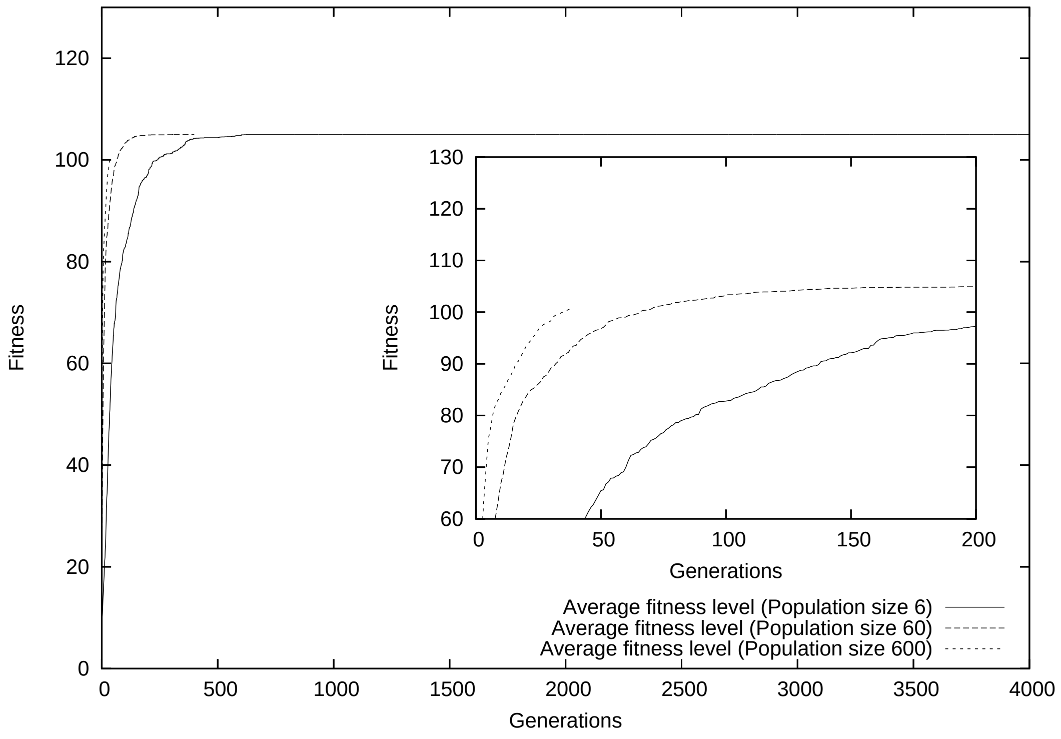

Figure B.11: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against generations

The graph in Figure B.11 shows the average fitness of all three population sizes. They are shown against generations to show the difference in the number of generations the different population sizes can evolve for in the same length of time. While the higher population sizes do better to start with the population size of 6 manages to improve its fitness to the same level as the other population sizes with the extra generations it can achieve in the time available to it.

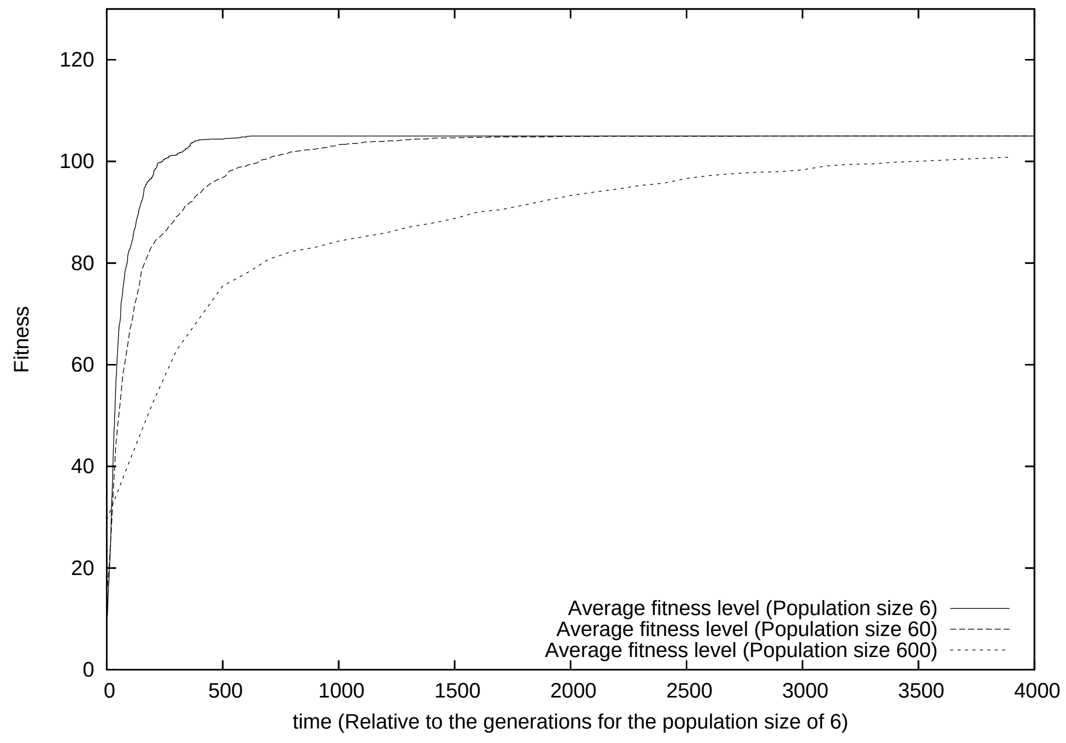

Figure B.12: Average Fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph in Figure B.12 shows the average fitness of all three population sizes. They are shown against the time taken to run the GA. This graph shows that the population size of 6 actually reaches the level that all the population sizes get stuck on first, while the higher population sizes take longer to get there.

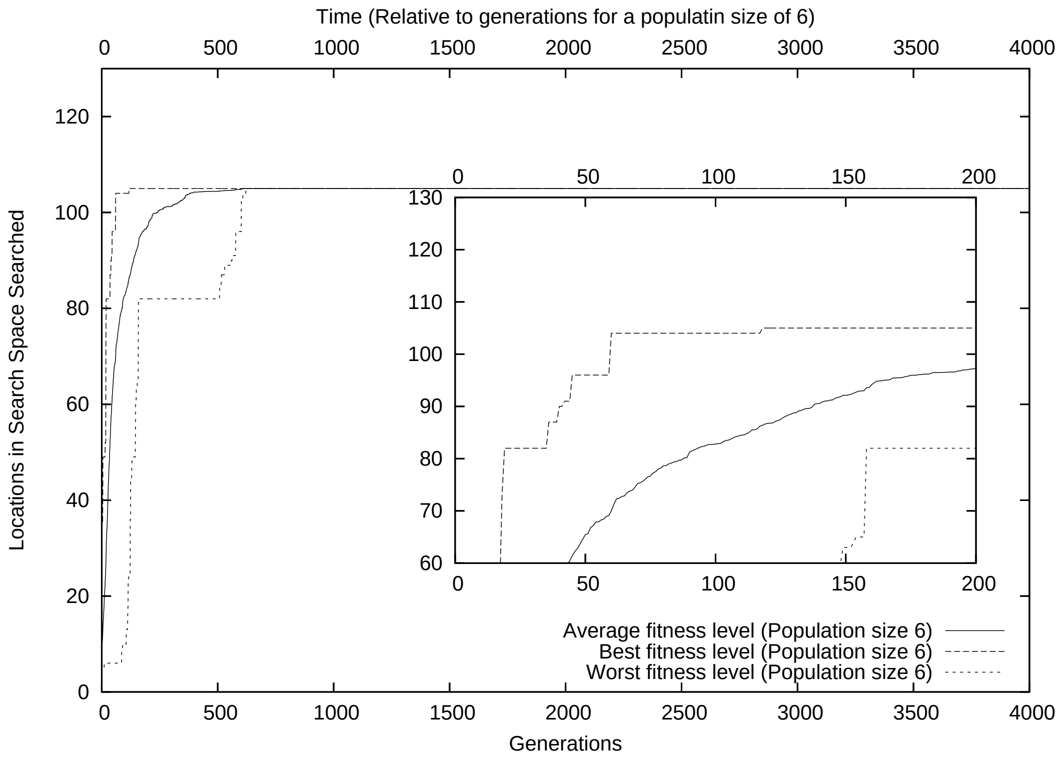

Figure B.13: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 6

The graph in Figure B.13 shows the average fitness level of the GA with a population size of 6, the best fitness level the GA achieved with a population size of 6 and the worst best fitness level the GA achieved with a population size of 6.

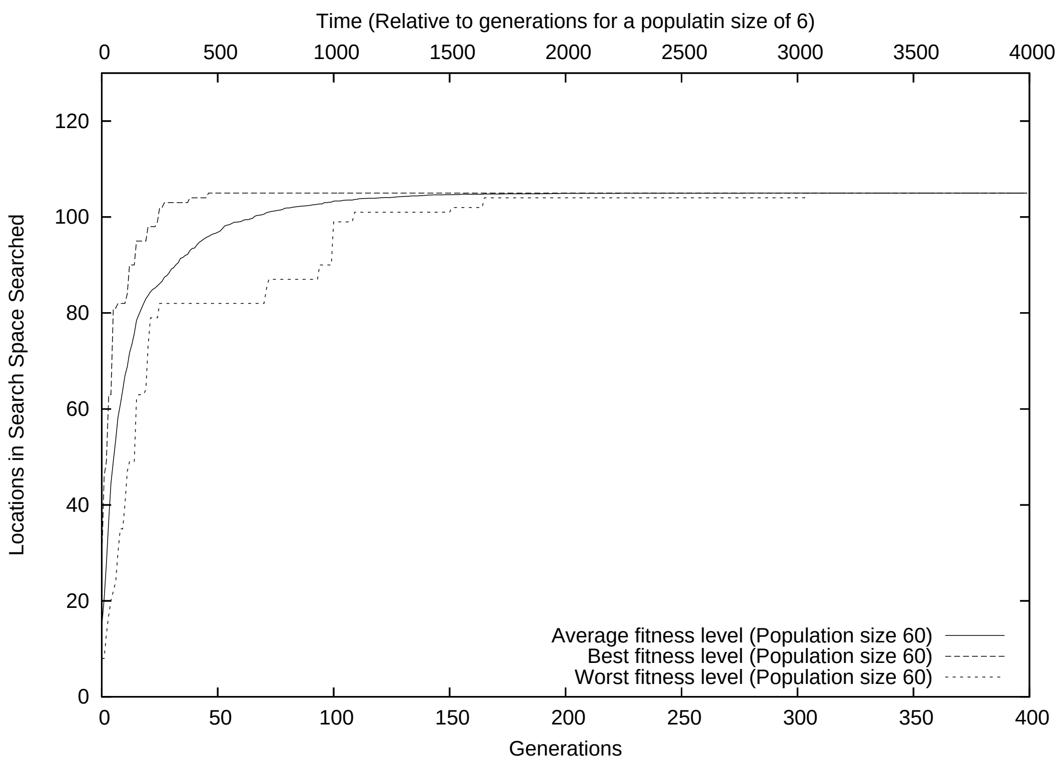

Figure B.14: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 60

The graph in Figure B.14 shows the average fitness level of the GA with a population size of 60, the best fitness level the GA achieved with a population size of 60 and the worst best fitness level the GA achieved with a population size of 60.

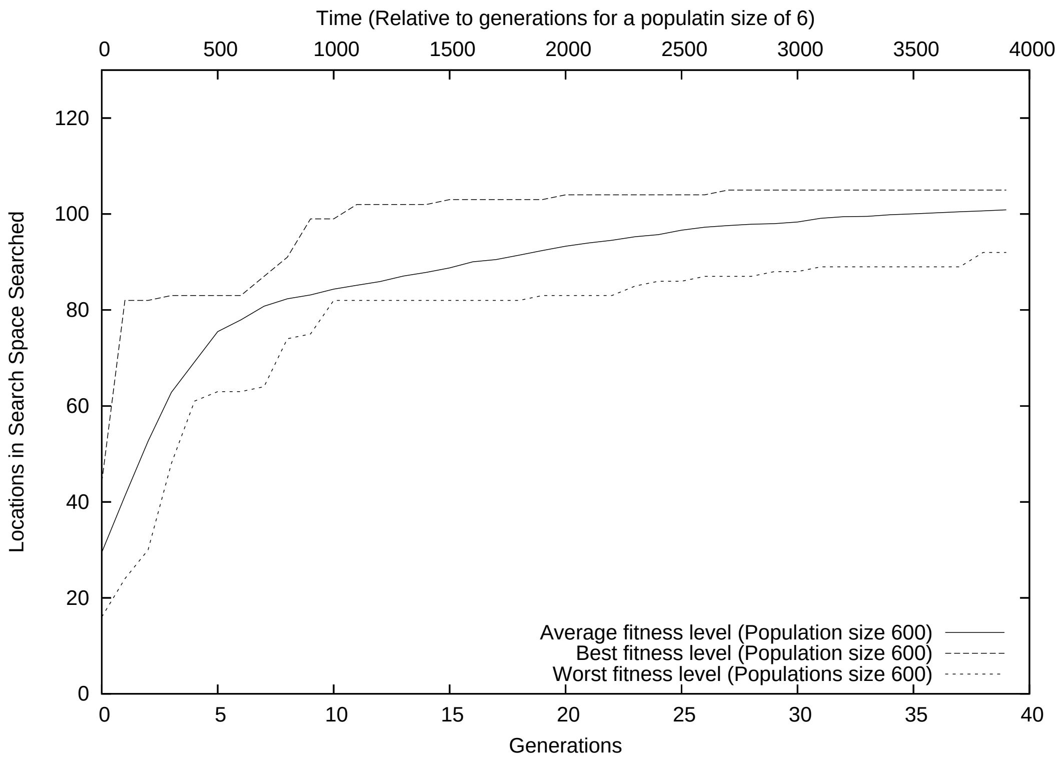

Figure B.15: Fitness levels, of the best of each of 50 runs, for the GA with a population size of 600

The graph in Figure B.15 shows the average fitness level of the GA with a population size of 600, the best fitness level the GA achieved with a population size of 600 and the worst best fitness level the GA achieved with a population size of 600.

Population Size	6	60	600
40 Generations	56	93	100
400 Generations	104	105	—
4000 Generations	105	—	—

Table B.3: Average fitness levels, of the best of each of 50 runs, of the three different population sizes tested

Table B.3 shows the fitness levels achieved by the GA with the three different population sizes after 40, 400 and 4000 generations. After 40 generations the GAs with a population size of 60 and 600 have achieved the same level of fitness. The population size of 6 has achieved an almost equal level of fitness.

The GA with a population size of 600 has used up all the time available to it by the 40^{th} generation so never manages to improve on the fitness level of 100. Both the GAs with a population size 6 and 60 had achieved the same level of fitness by their 400^{th} generation.

The GA with a population size of 60 has used up all the time available to it by its 400^{th} generation so never manages to improve its fitness level above 105. The GA with a population size of 6 failed to achieve a higher level of fitness despite having enough time to run for 4000 generations.

B.2 Population Sizes and their Effects on Performance Using a Different Model of Elitism

This section shows and discusses the results of the experiments (see section 3.4.3) testing the effects of population sizes on the performance of the GA when a different model of elitism is used.

B.2.1 One Max Problem

The results of the experiments on the One Max Problem are shown in Figure B.16 and Table B.4. The graph in Figure B.16 shows the average best fitness levels for the population sizes against time. Table B.4 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

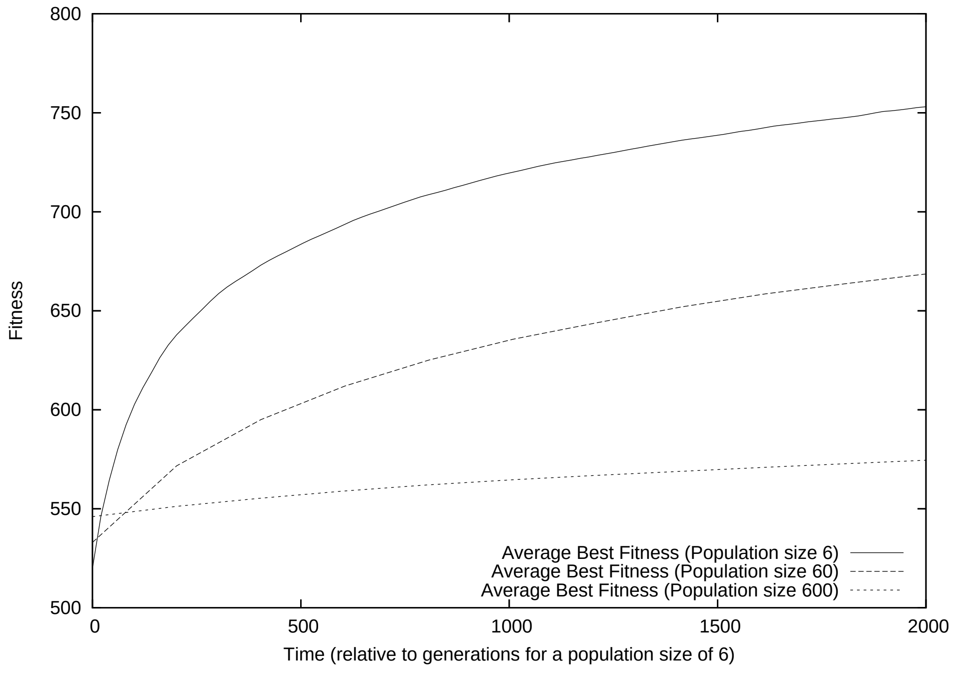

Figure B.16: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph shown in Figure B.16 shows that for the One Max Problem a population size 6 outperforms a population size of 60 and 600. Compared to the results shown in Figure B.2 the population sizes of 60 and 600 have performed better, but they are still a good distance behind the results for the population size of 6.

Population Size	6	60	600
40 Generations	557	595	590
400 Generations	676	701	–

Table B.4: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table B.4 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. In Table B.1 at 400 generations the population size of 6 had achieved a higher average best fitness level than that of the population size of 60 at 400 generations. This configuration of GA which carries the best $\frac{1}{6}$ forward to the next population showed a similar anomaly, where the population size of 60 has a higher average best fitness level than the population size of 600 at 40 generations. This suggests that the anomaly is related to the elitism levels the GAs are using.

B.2.2 Deceptive Trap Functions

The results of the experiments on the Deceptive Trap Functions are shown in Figure B.17 and Table B.5. Figure B.17 shows the average best fitness levels for the population sizes against time, while Table B.5 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

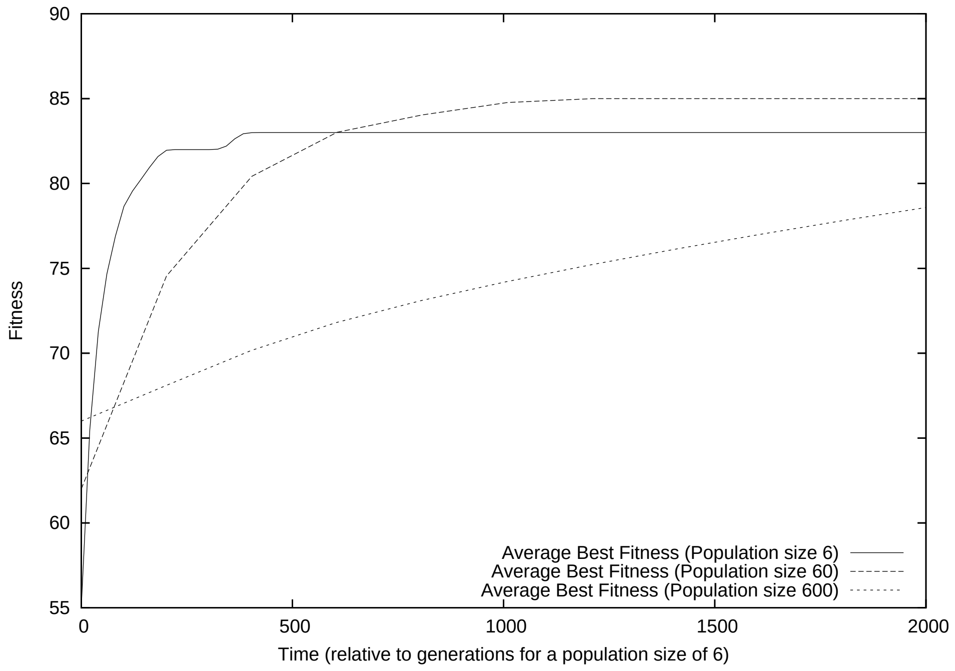

Figure B.17: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

Figure B.17 shows that for the GA tackling the Deceptive Trap Functions The population size of 60 outperforms the population sizes of 6 and 600. The population size of 6 performs better than the population size of 60 in the early stages, but as the time the GA has been evolving for increases the population size of 60 overtakes it.

Again compared to the results shown in Figure B.7 the population sizes of 60 and 600 have performed better. Though the population size of 600 still does not perform as well as the population sizes of 6 and 60.

Population Size	6	60	600
40 Generations	72	81	83
400 Generations	83	85	–

Table B.5: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table B.5 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. Section B.2 shows an anomaly where the population size of 6 had achieved a higher average best fitness level than the population size of 60. This anomaly is not present in the results for this configuration of GA.

B.2.3 GA Hard Problem

The results of the experiments on the GA Hard Problem are shown in the graph in Figure B.18 and Table B.6. The graph in Figure B.18 shows the average best fitness levels for the population sizes against time, while Table B.6 shows the average best fitness level achieved by each of the population sizes at 40 and 400 generations.

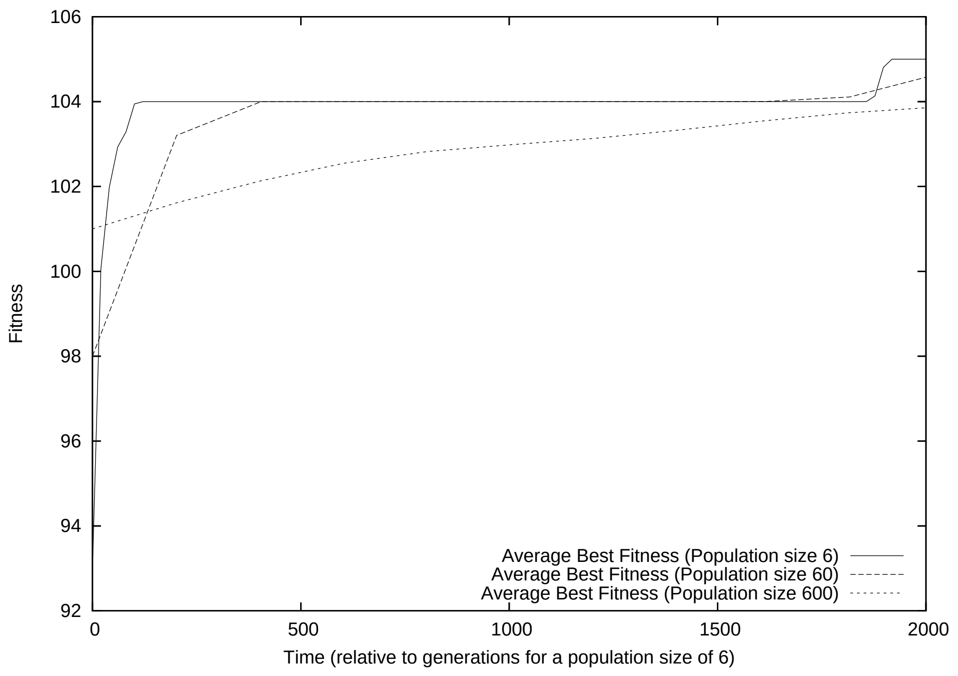

Figure B.18: Average fitness levels, of the best of each of 50 runs, for the three different population sizes, shown against time

The graph shown in Figure B.18 shows that for the GA Hard Problem the population sizes of 6 and 60 are very close. The population size of 6 outperforms the population size of 60 and 600 in the time available to it.

Compared to the results shown in Figure B.12 the results are similar with the exception of the population size of 600 which has performed better, though still not as well as the population sizes of 6 and 60.

Population Size	6	60	600
40 Generations	102	104	104
400 Generations	104	105	–

Table B.6: Average fitness levels, of the best of 50 runs of the three population sizes tested

Table B.6 shows the average best fitness levels achieved by the different population sizes at 40 and 400 generations. Interestingly at 40 generation the population sizes of 60 and 600 have achieved the same average best fitness level.

B.3 Population Sizes and their Effects on Efficiency

This section shows and discusses the results of the experiments (see section 3.4.4) testing the effects of population sizes on the efficiency of the GA.

B.3.1 One Max Problem

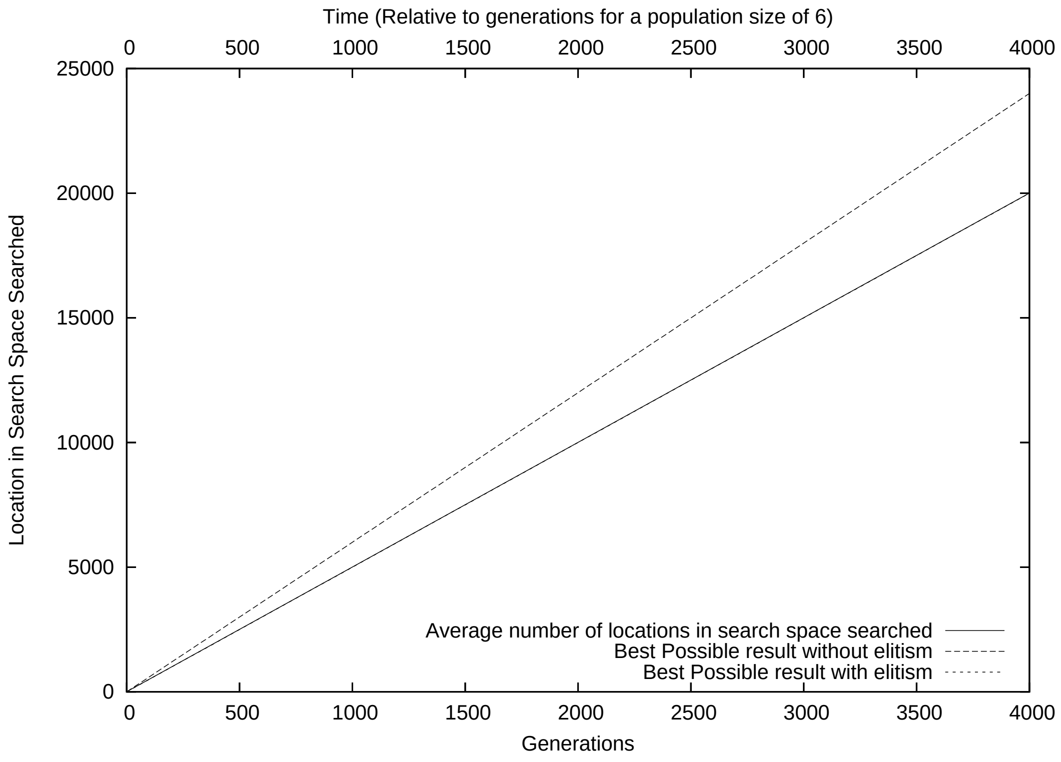

Figure B.19: Number of locations in search space searched by the GA with a population size of 6

The graph in Figure B.19 shows the average number of locations in search space that the GA with a population size of 6 searched. It also shows the best possible result for a GA without elitism and the best possible result for a GA using an elitism level of one.

The GA with a population size of 6 has managed to achieve very close to the best efficiency that it could. The best possible result is not available to the GA as it is using elitism which results in individuals being carried forward and as such the next generation will always contain some individuals that have been seen before.

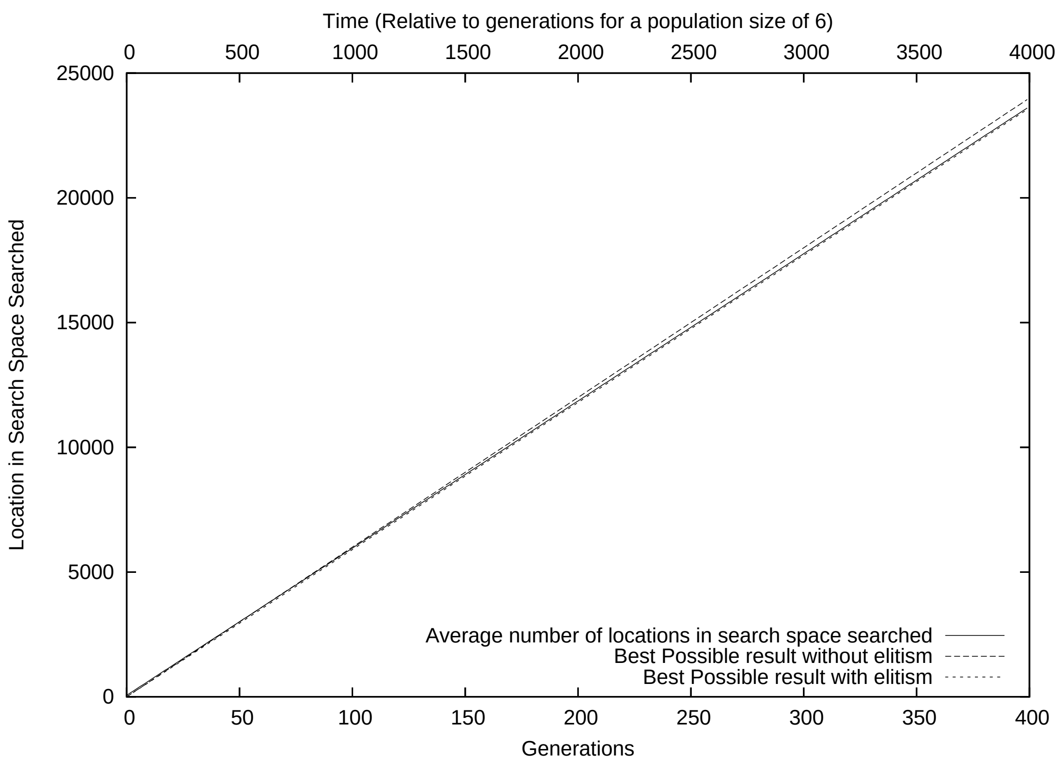

Figure B.20: Number of locations in search space searched by the GA with a population size of 60

The graph in Figure B.20 shows the average locations in search space that the GA with a population size of 60 searched. The best possible result for a GA without elitism and the best possible result for a GA using elitism are also shown.

The GA with a population size of 60 has managed to achieve a result very close to that of the best possible results for a GA with elitism. Due to the lower number of generations that the GA with a population size of 60 can run in the available time, the difference between the best possible results and the best possible result for a GA with an elitism level of one is less than the difference for the GA with a population size of 6.

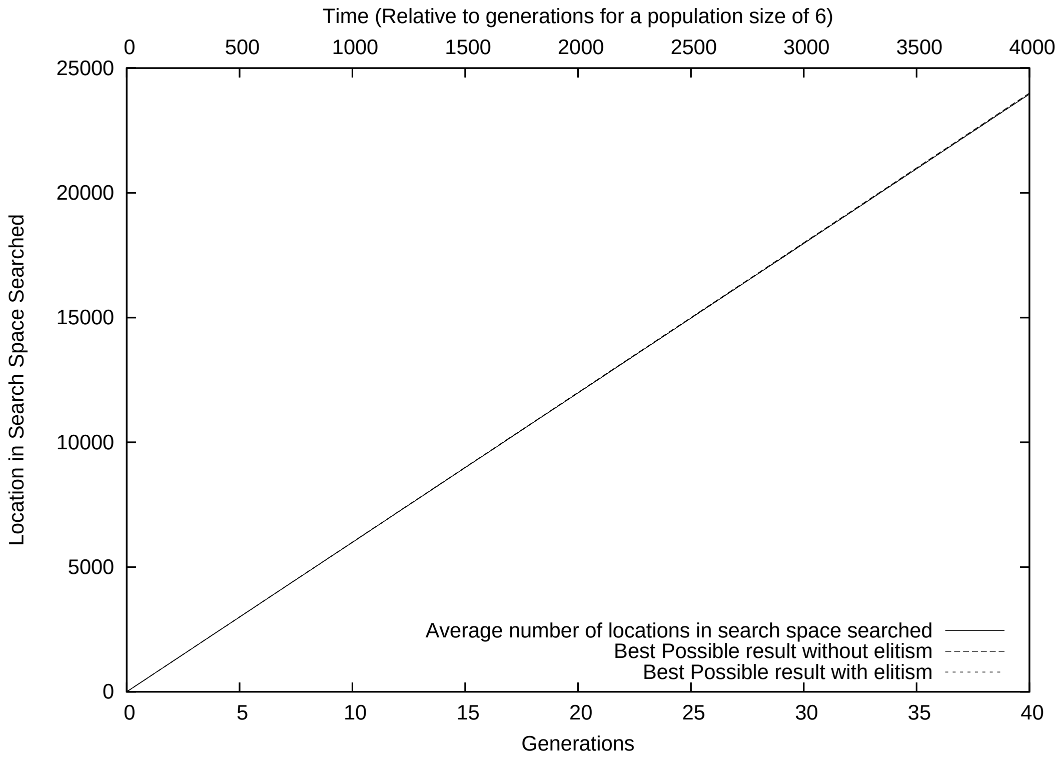

Figure B.21: Number of locations in search space searched by the GA with a population size of 600

The graph in Figure B.21 shows the average number of locations in search space that the GA with a population size of 600 has searched. The best possible result for a GA without elitism and the best possible result for a GA using elitism are also shown.

The GA with a population size of 600 has managed to achieve an almost perfect result for the efficiency. Again the lower number of generations has helped to improve the efficiency as the elitism has only carried forward a total of 40 individuals over the whole run.

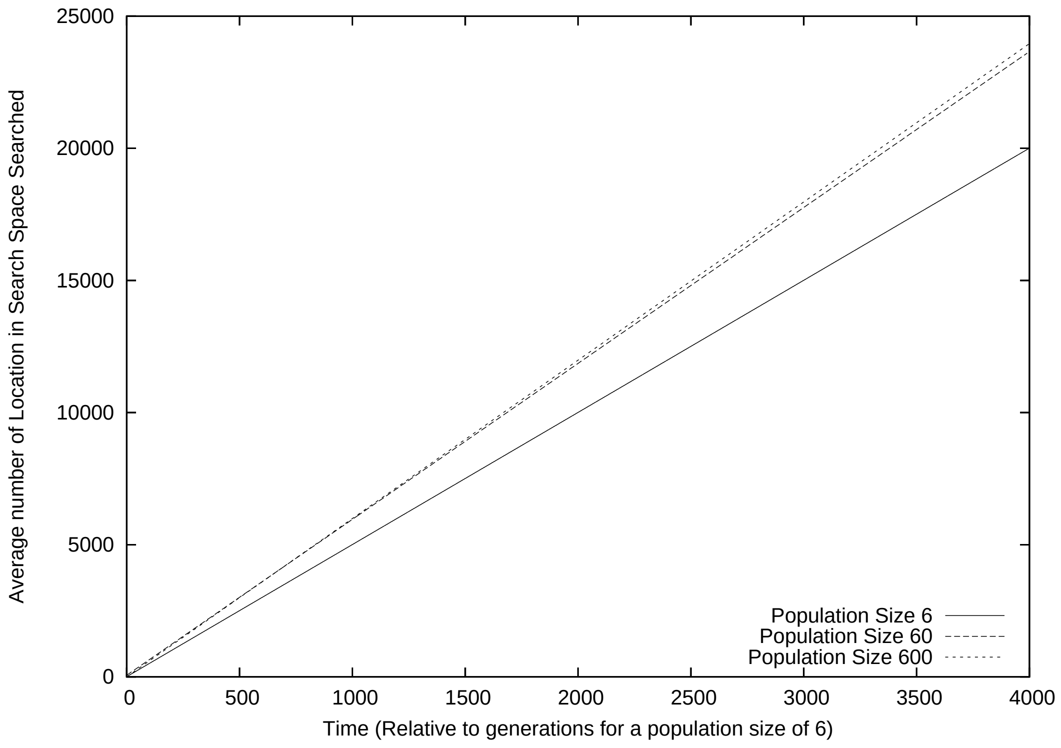

Figure B.22: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure B.22 shows the average number of locations in search space searched by the GAs using the three population sizes. This shows that the population size of 6 has the worst efficiency while the population size of 600 has the best. The difference between the efficiency of the population size of 60 and the efficiency of the population size of 600 is smaller than that of the difference between the the population size of 6 and 60.

B.3.2 Deceptive Trap Functions

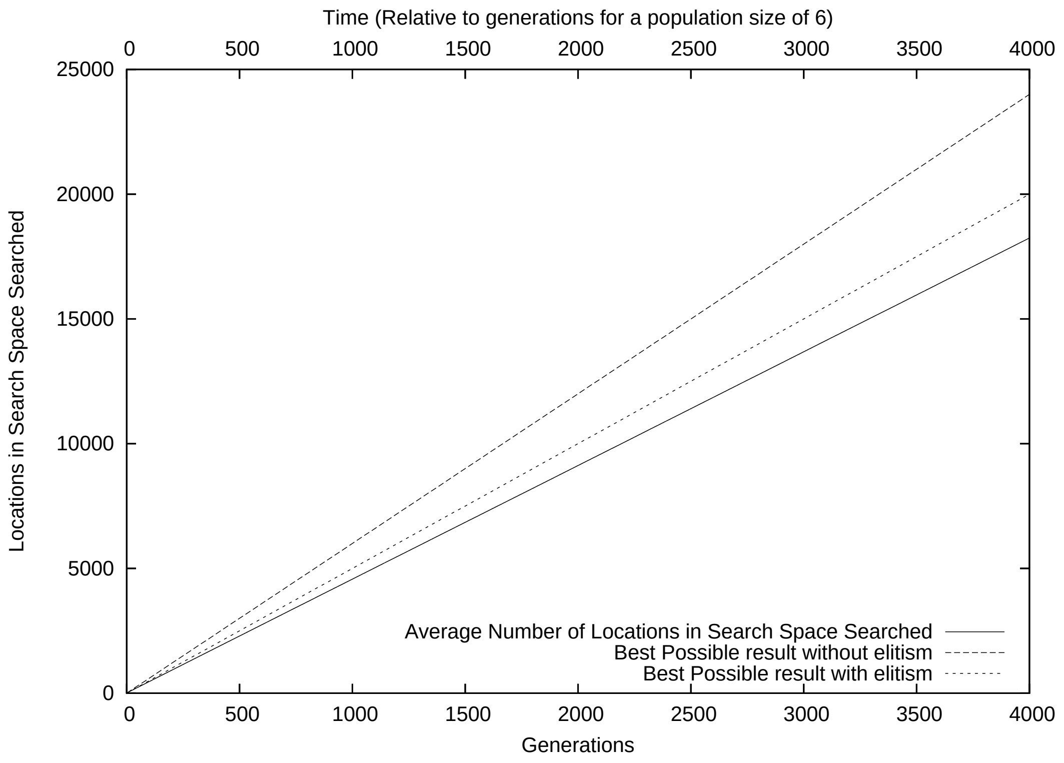

Figure B.23: Number of locations in search space searched by the GA with a population size of 6

The graph in Figure B.23 shows the average number of locations in search space that the GA with a population size of 6 has searched. The best possible result for a GA without elitism and the best possible result for a GA using elitism are also shown.

Compared to the results for the GA tackling the One Max Problem a gap has appeared, between the best possible results for a GA using elitism and the average result for the GA with a population size of 6. This shows that there is room for improvement in the efficiency of the GA tackling the Deceptive Trap Functions with a population size of 6.

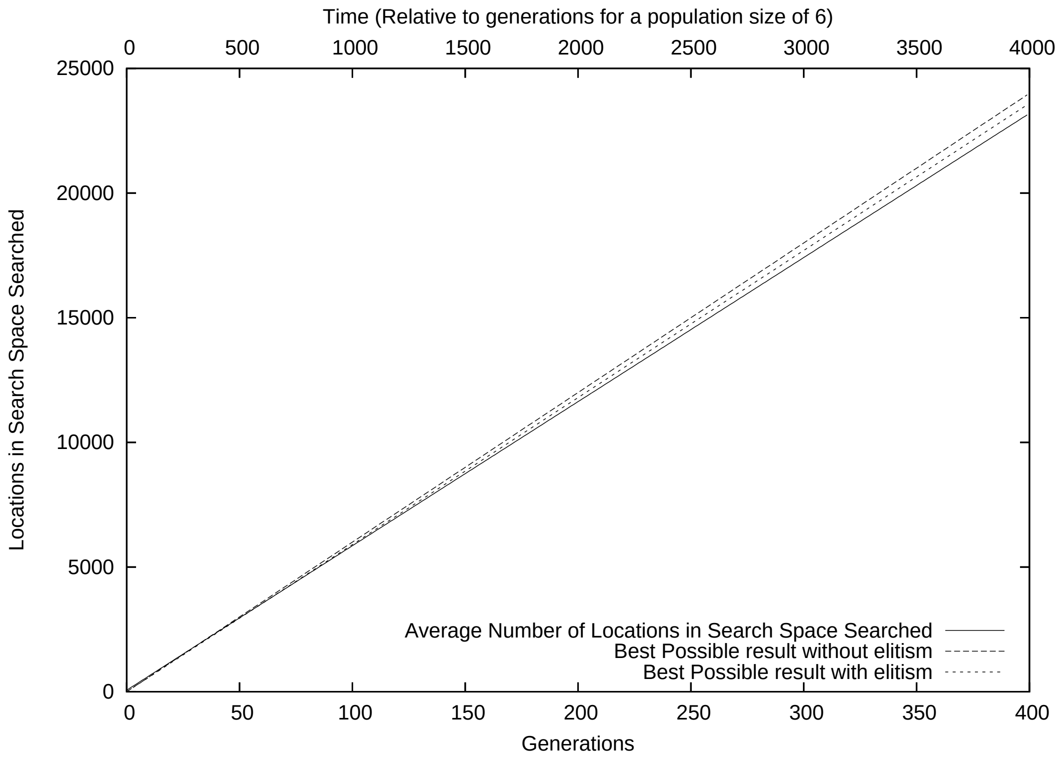

Figure B.24: Number of locations in search space searched by the GA with a population size of 60

The graph in Figure B.24 shows the average number of locations in search space that the GA with a population size of 60 has searched. The best possible results for a GA without elitism and a GA using elitism are also shown.

Again when compared to the results of the GA tackling the One Max Problem a gap has appeared, between the best possible result for a GA using elitism and the average result for the GA with a population size of 60. While the possible improvement available to the efficiency of the GA is not as large as that of the GA using a population size of 6, there is still room for improvement.

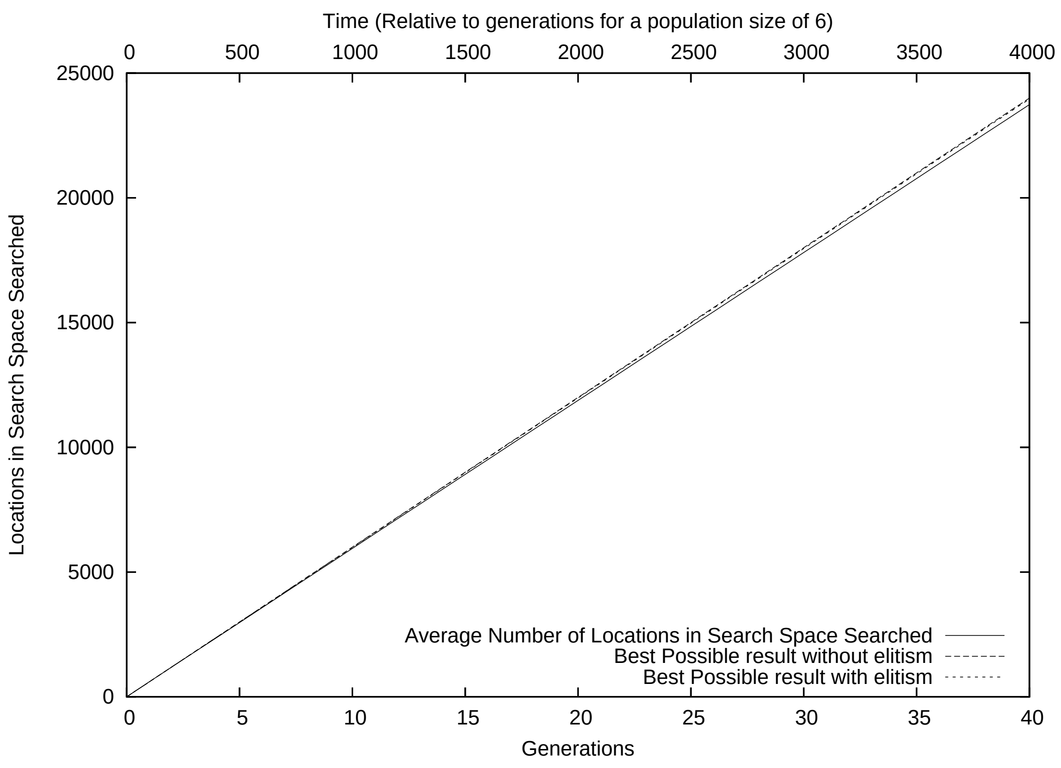

Figure B.25: Number of locations in search space searched by the GA with a population size of 600

The graph in Figure B.25 shows the average number of locations in search space that the GA with a population size of 600 has searched. The best possible results for a GA without elitism and a GA using elitism are also shown.

The GA using a population size of 600 shows a bit more room for improvement, in the efficiency of the GA, than the GA tackling the One Max Problem did.

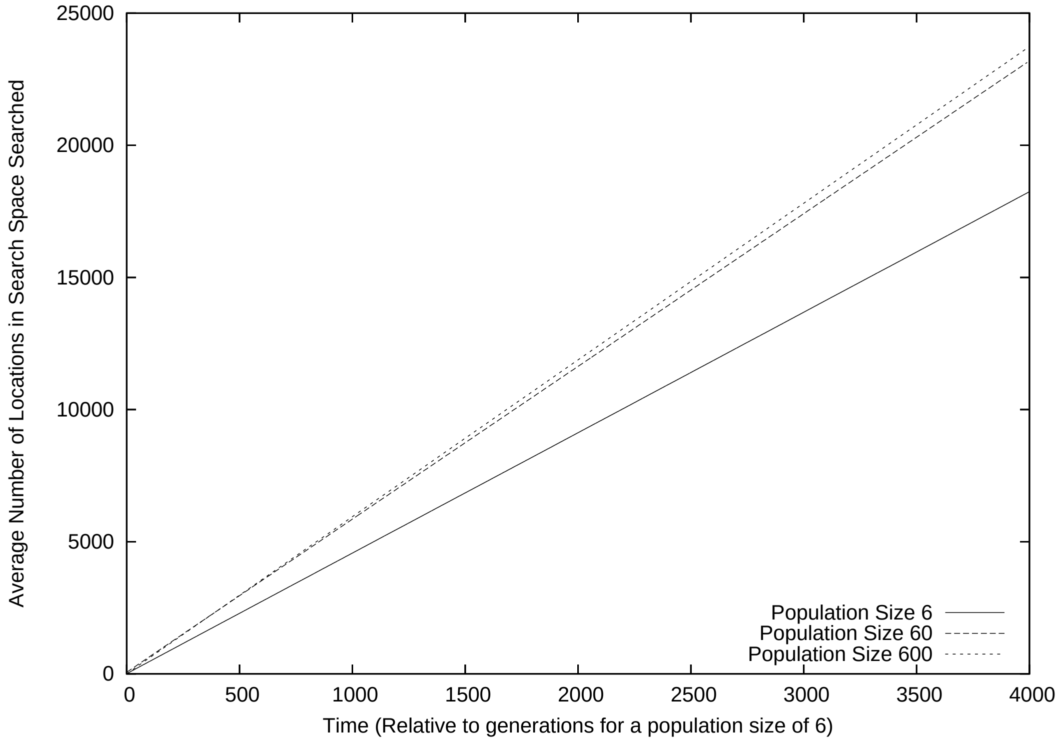

Figure B.26: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure B.26 shows the average number of locations in search space searched by the GAs using the three population sizes. A similar result is shown to that of Figure B.22 with the difference in efficiency between the GA using a population size of 6 and the GA using a population size of 60 being larger than the difference between the GA using a population size of 60 and the GA using a population size of 600.

B.3.3 GA Hard Problem

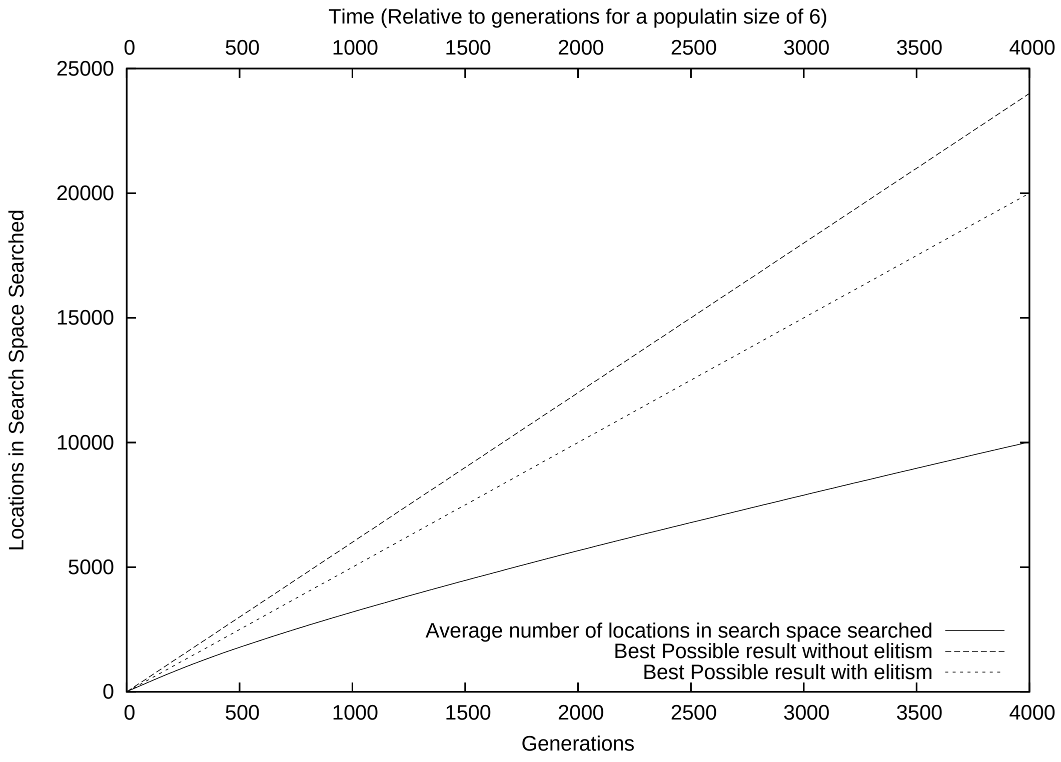

Figure B.27: Number of locations in search space searched by the GA with a population size of 6

The graph in Figure B.27 shows the average number of locations in search space that the GA with a population size of 6 has searched. The best possible results for a GA without elitism and a GA using elitism are also shown.

Compared to the results from the GAs tackling the One Max Problem and the Deceptive Trap Functions the efficiency of the GA has decreased by a large amount, as shown by the gap between the 'Best Possible result with elitism' and the 'Average number of locations in search space searched' lines.

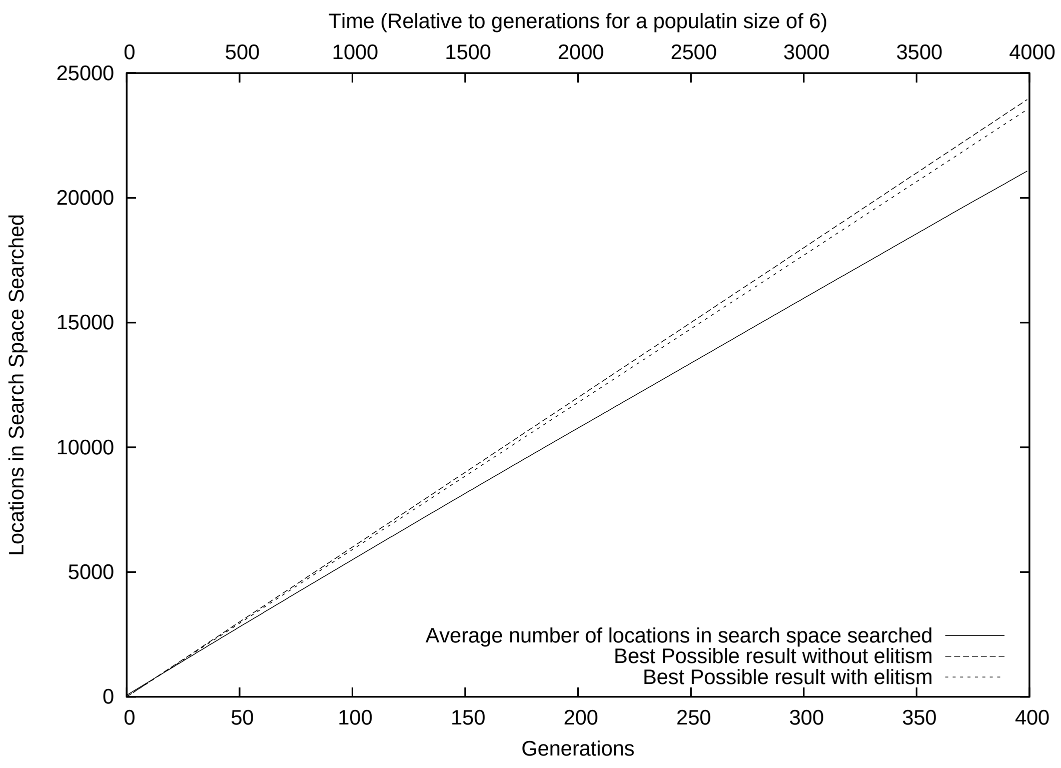

Figure B.28: Number of locations in search space searched by the GA with a population size of 60

The graph in Figure B.28 shows the average number of locations in search space that the GA with a population size of 60 has searched. The best possible results for a GA without elitism and a GA using elitism are also shown.

Again the gap between the best possible result for a GA using elitism and the actual result achieved by the GA tackling the GA Hard Problem has grown compared to the One Max Problem and the Deceptive Trap Functions. Though the gap is still smaller for the GA using a population size of 60 than that of the one using a population size of 6.

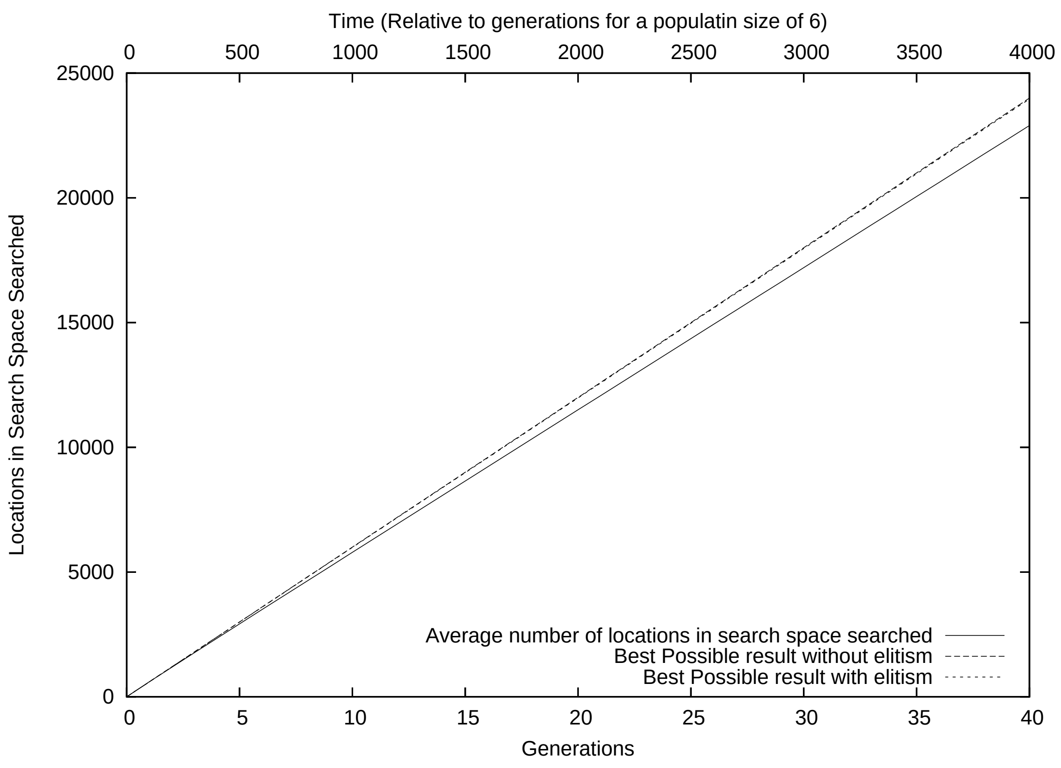

Figure B.29: Number of locations in search space searched by the GA with a population size of 600

The graph in Figure B.29 shows the average number of locations in search space that the GA with a population size of 600 has searched. The best possible results for a GA without elitism and a GA using elitism are also shown.

Again the gap between the best result for a GA using elitism and that achieved by the GA using a population size of 600 has grown when compared to the One Max Problem and the Deceptive Trap Functions.

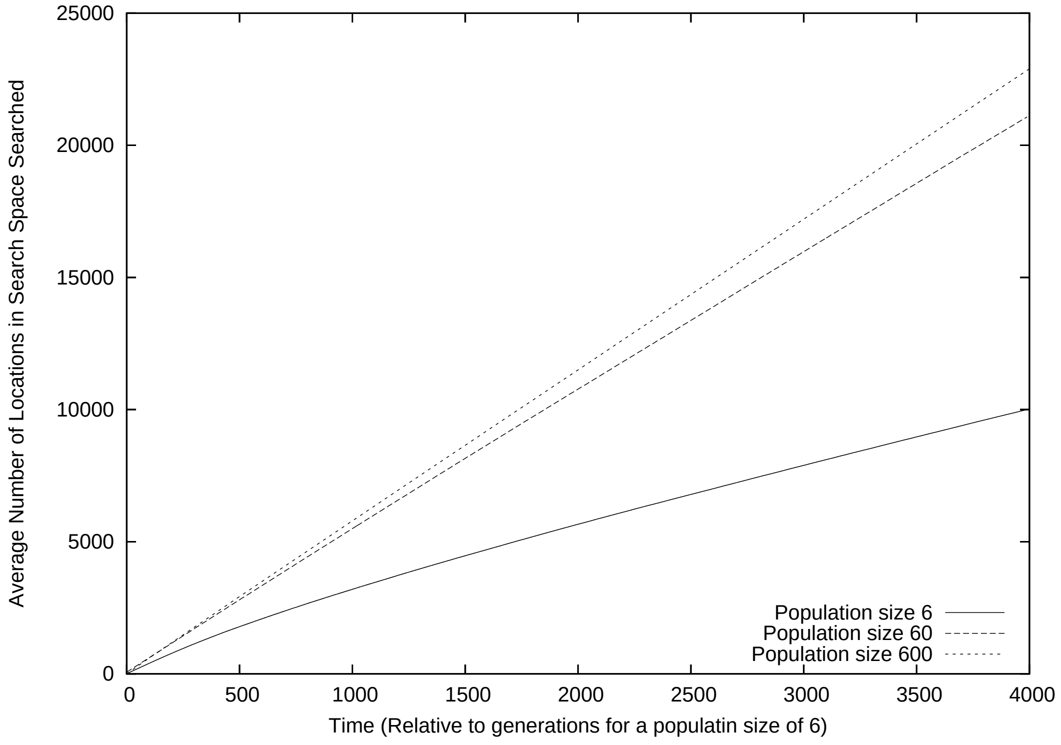

Figure B.30: Number of locations in search space searched by the GA with each of the population sizes tested

The graph shown in Figure B.30 shows the average number of locations in search space searched by the GAs using the three population sizes. Again the difference between the efficiency of the GA using a population size of 6 and the GA using a population size of 60 is larger than the difference in efficiency between the GA using a population size of 60 and the GA using a population size of 600.

B.4 Elitism Levels and their Effects on Performance

This section shows and discusses the results of the experiments (see section 3.4.5) the effects of elitism on the performance of GAs.

B.4.1 One Max Problem

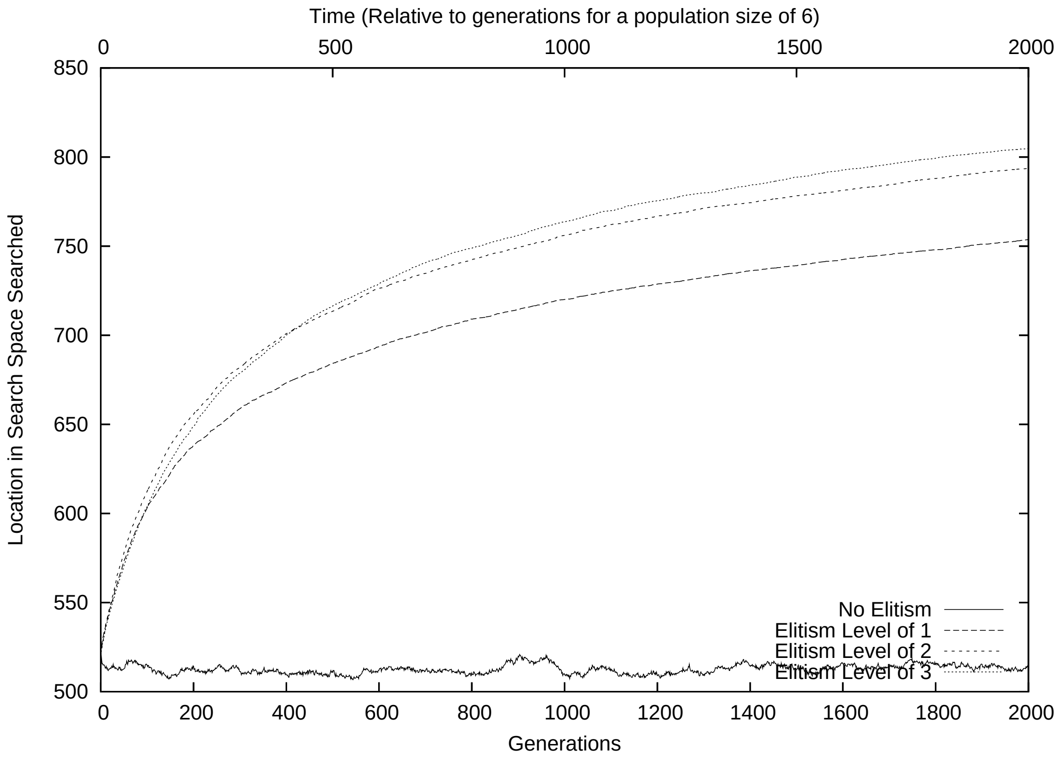

Figure B.31: Average Fitness Levels for the One Max Problem

The graph in Figure B.31 shows that for the One Max Problem the higher level of elitism is better. With a population size of 6 and no elitism the GA found it very hard to improve. With only one individual being carried forward the GA found it a lot easier to make improvements. With two individuals being carried forward the GA found it even easier to make improvements. The best setting for elitism with the one max problem was with three individuals being carried forward.

B.4.2 Deceptive Trap Functions

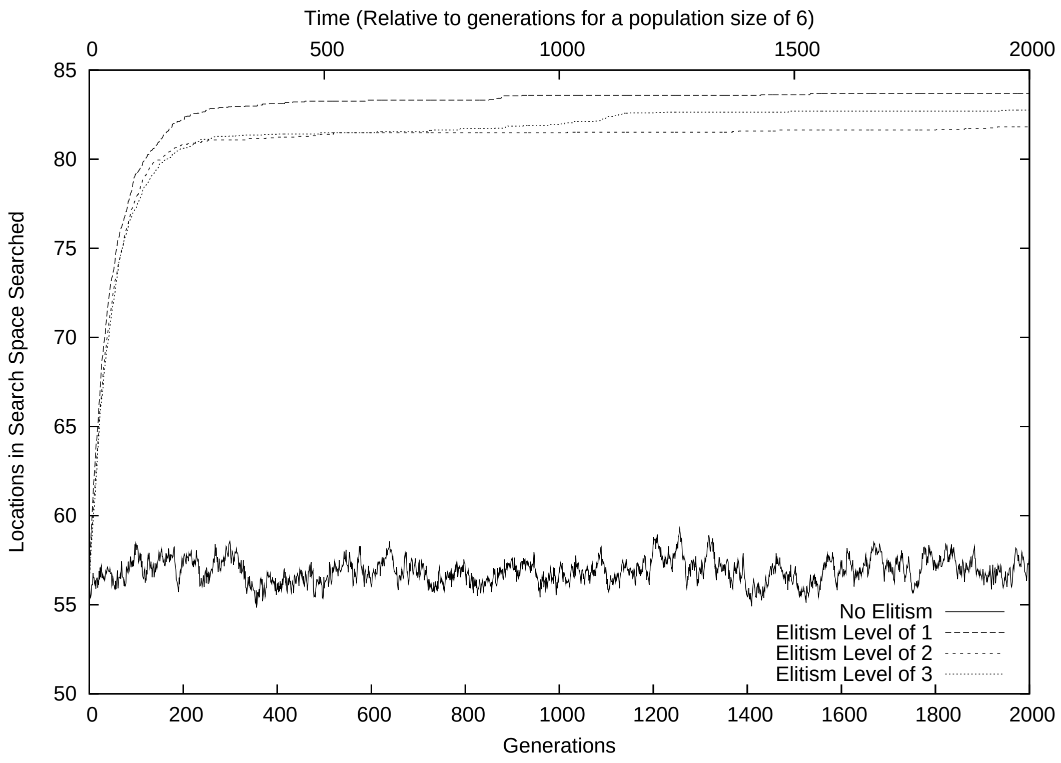

Figure B.32: Average Fitness Levels for the Deceptive Trap Function

Figure B.32 shows that for the Deceptive Trap Function, being tackled by the GA with a population size of 6, the best level of elitism is to carry one individual forward to the next generation. Carrying two individuals and three individuals forward produces good results but not as good as just carrying one forward, though the difference in fitness between the three levels of elitism is marginal. Again the GA using no elitism found it hard to improve.

B.4.3 GA Hard Problem

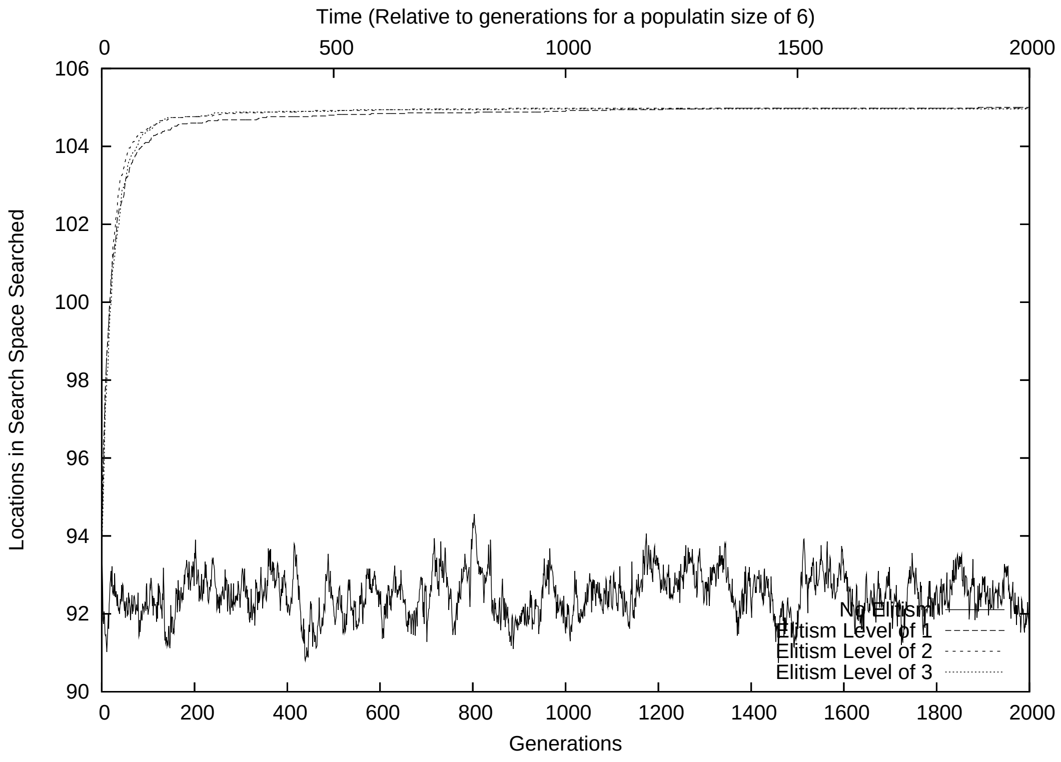

Figure B.33: Average Fitness Levels for the GA Hard Problem

The graph shown in Figure B.33 shows that for the GA Hard Problem carrying two individuals forward to the next generation gives the best result. Carrying one or three individuals forward gives a similar result to that of carrying two individuals forward. Carrying one, two or three individuals forward results in an almost identical level of fitness after 2000 generations. Again when the GA is not using elitism it fails to keep improvements in the level of fitness.

Appendix C

Full Results for Chapter 5

C.1 One Max Problem

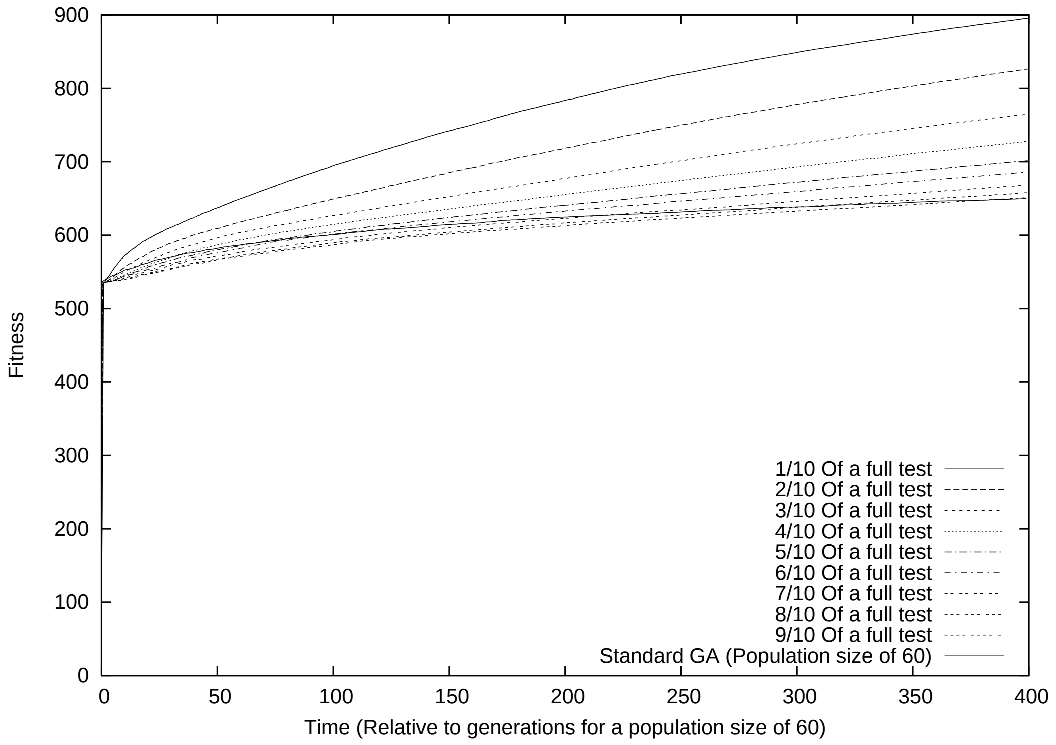

Figure C.1: Results of using a partial fitness function on the One Max Problem, replacing $\frac{1}{10}$ the population after each partial fitness test

The graph in Figure C.1 shows that when replacing $\frac{1}{10}$ of the population after each generation the best partial fitness function is one that tests $\frac{1}{10}$ of an individual each generation. The next best partial fitness function is one that tests $\frac{2}{20}$ of an individual each generation. This pattern continues with the next best being a partial fitness function that test tests $\frac{3}{10}$ of an individual each generation, and then one that tests $\frac{4}{10}$ of an individual each generation. The partial fitness function that tests $\frac{9}{10}$ of an individual each generation gives very similar results to a standard GA.

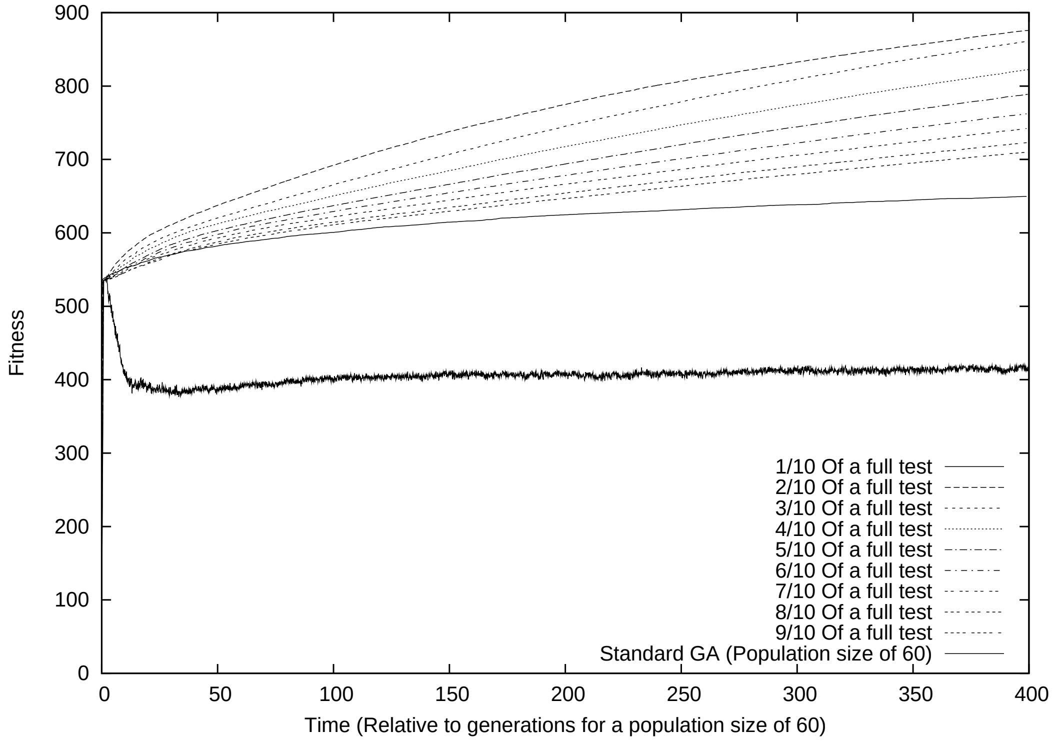

Figure C.2: Results of using a partial fitness function on the One Max Problem, replacing $\frac{2}{10}$ the population after each partial fitness test

The graph in Figure C.2 shows that when replacing $\frac{2}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{2}{10}$ to $\frac{9}{10}$

of an individual each generation, is greater than for the same fitness functions when replacing $\frac{1}{10}$ of the population. The results of the partial fitness function that tests $\frac{1}{10}$ of an individual each generation performs very badly.

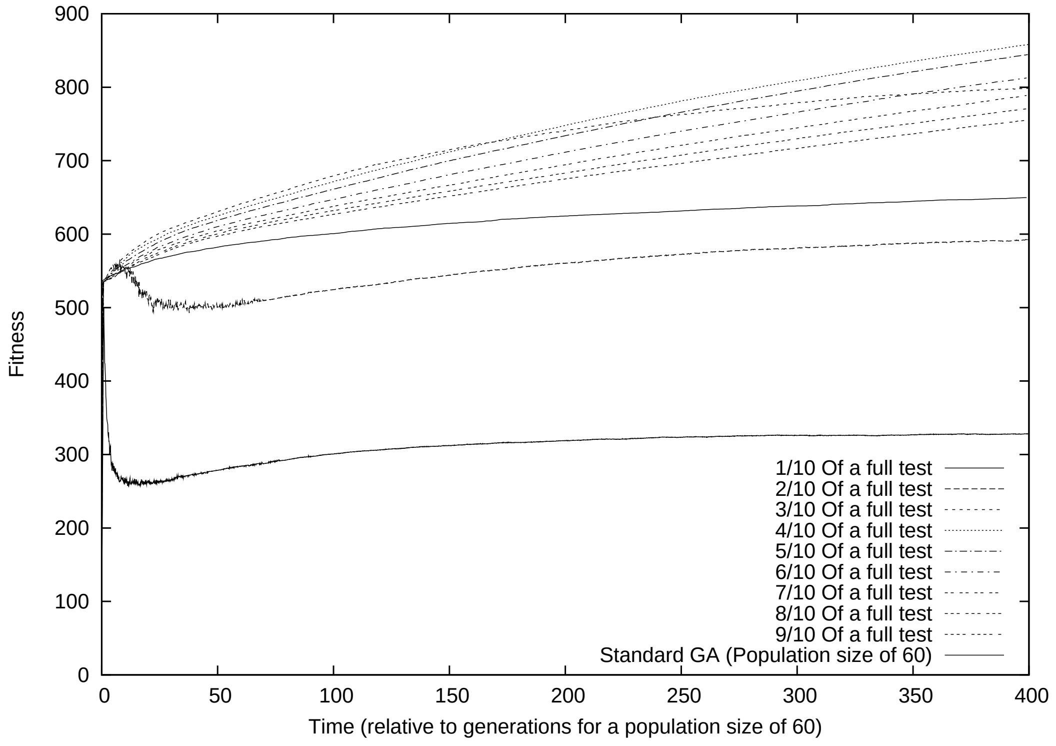

Figure C.3: Results of using a partial fitness function on the One Max Problem, replacing $\frac{3}{10}$ the population after each partial fitness test

The graph in Figure C.3 shows that when replacing $\frac{3}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{4}{10}$ to $\frac{9}{10}$ of an individual each generation, is greater than for the same fitness functions when replacing $\frac{3}{10}$ of the population. The partial fitness function that tests $\frac{3}{10}$ of an individual each generation does not perform as well as it did in the previous experiments, but it still performs better than a standard GA. The partial fitness functions that test $\frac{1}{10}$ and

$\frac{2}{10}$ of an individual each generation performed very badly.

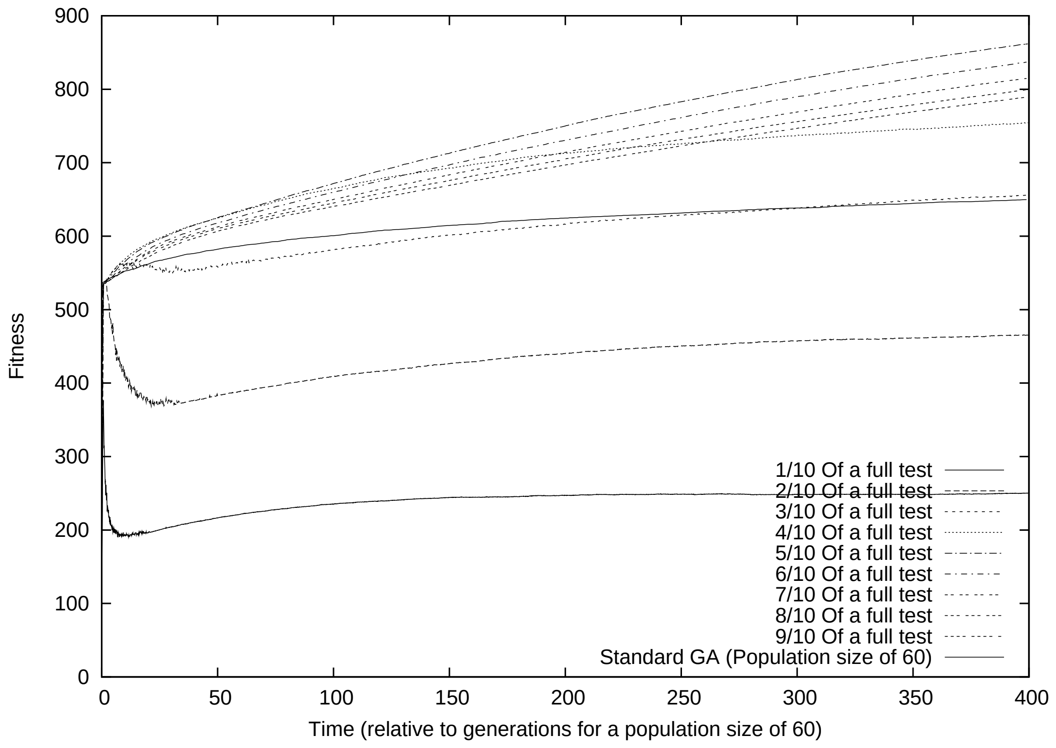

Figure C.4: Results of using a partial fitness function on the One Max Problem, replacing $\frac{4}{10}$ the population after each partial fitness test

The graph in Figure C.4 shows that when replacing $\frac{4}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, is greater than for the same fitness functions when replacing $\frac{3}{10}$ of the population. The partial fitness function that tests $\frac{4}{10}$ of an individual each generation does not perform as well as it did in the previous experiments, but still performs better than a standard GA. The results of the partial fitness function that tests $\frac{3}{10}$ of an individual each generation is very similar to a standard GA. The results of the partial fitness functions that test $\frac{1}{10}$ and $\frac{2}{10}$ of an individual each generation performs very badly.

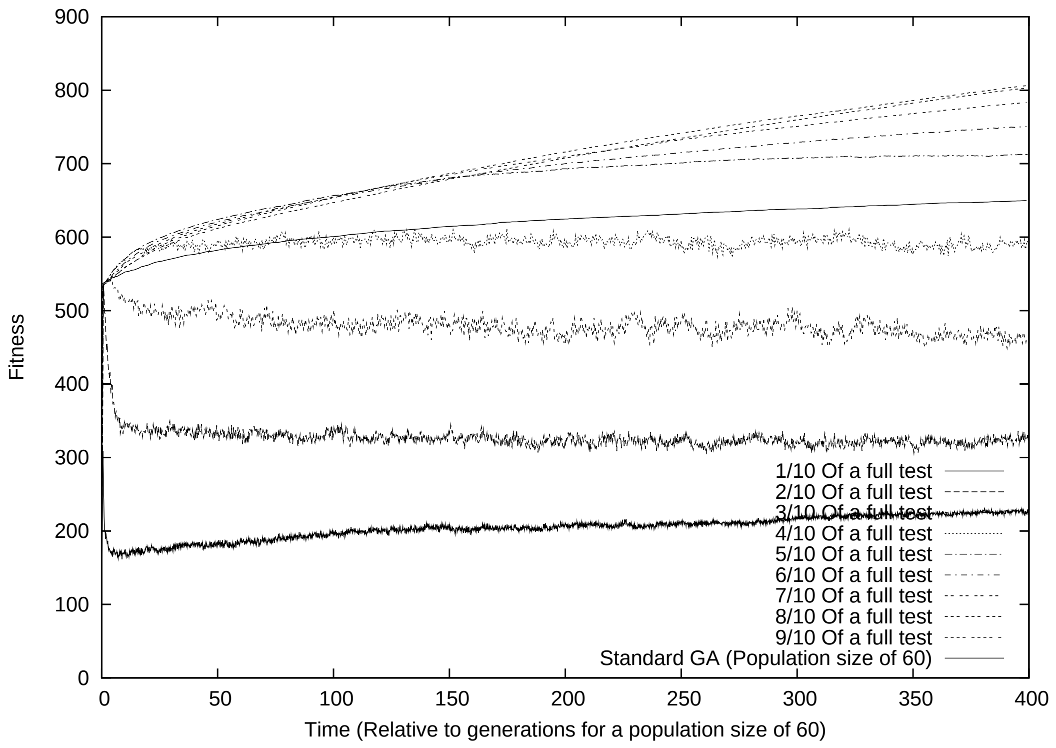

Figure C.5: Results of using a partial fitness function on the One Max Problem, replacing $\frac{5}{10}$ the population after each partial fitness test

The graph in Figure C.5 shows that when replacing $\frac{5}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, is lower than for the same fitness functions when replacing $\frac{4}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

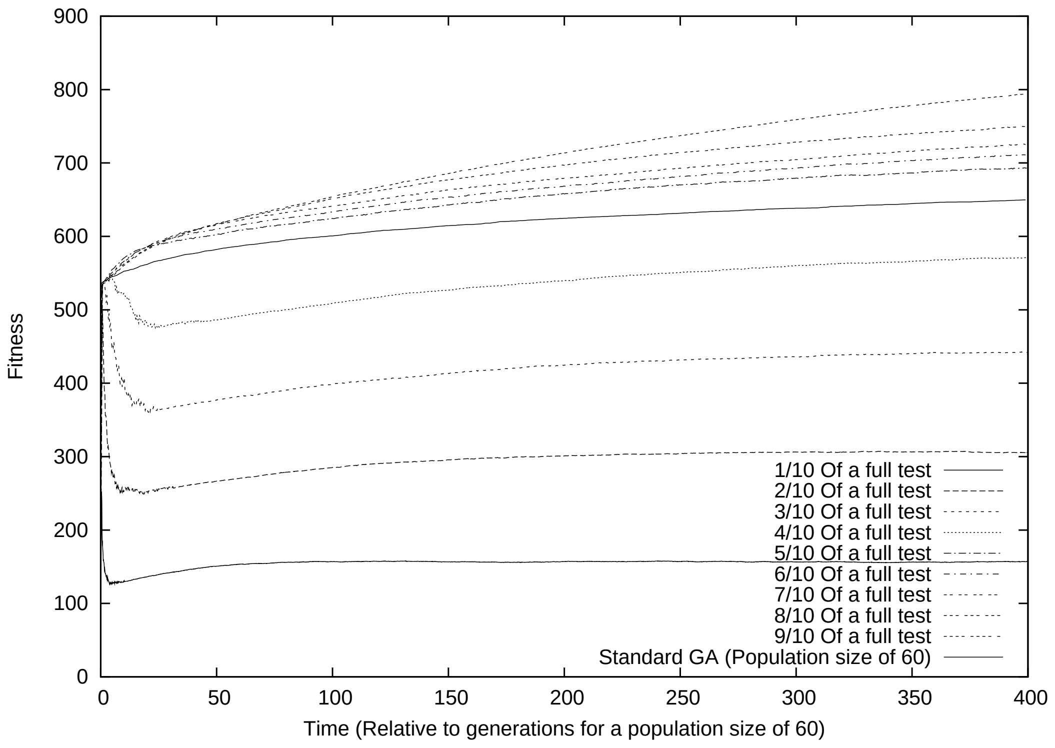

Figure C.6: Results of using a partial fitness function on the One Max Problem, replacing $\frac{6}{10}$ the population after each partial fitness test

The graph in Figure C.6 shows that when replacing $\frac{6}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, is lower than for the same fitness functions when replacing $\frac{5}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

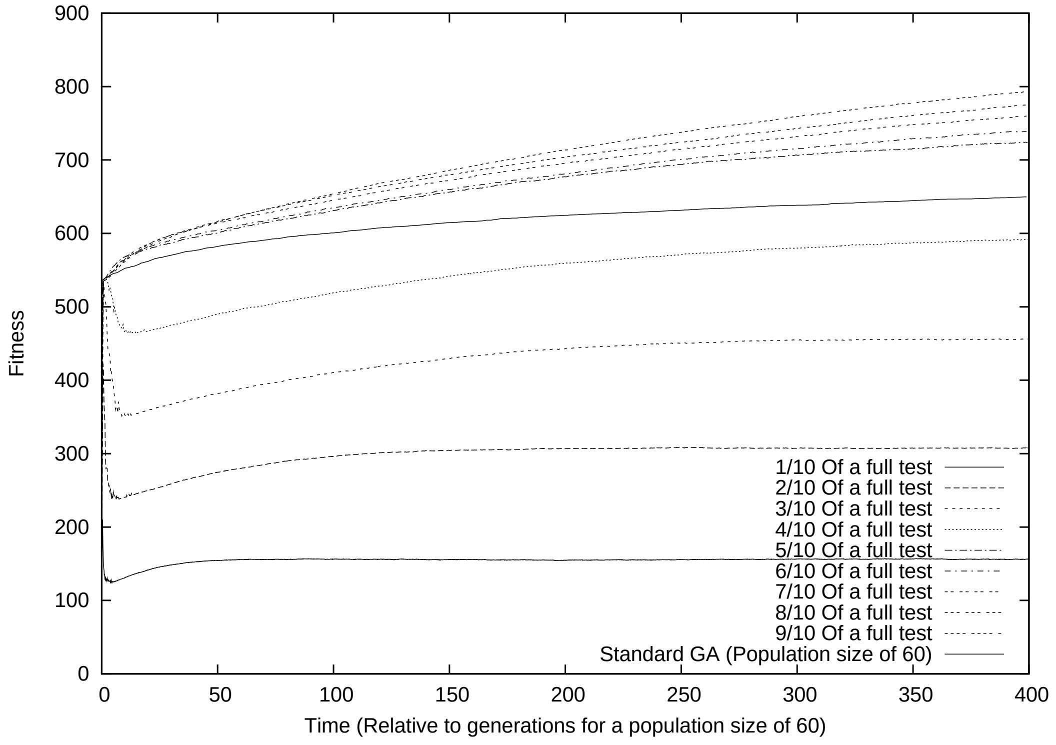

Figure C.7: Results of using a partial fitness function on the One Max Problem, replacing $\frac{7}{10}$ the population after each partial fitness test

The graph in Figure C.7 shows that when replacing $\frac{7}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{8}{10}$ of an individual each generation, performs slightly better than for the same fitness functions when replacing $\frac{6}{10}$ of the population. The partial fitness function that tests $\frac{9}{10}$ of an individual each generation performs very similar to the previous graph, where $\frac{6}{10}$ of the population were replaced each generation. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

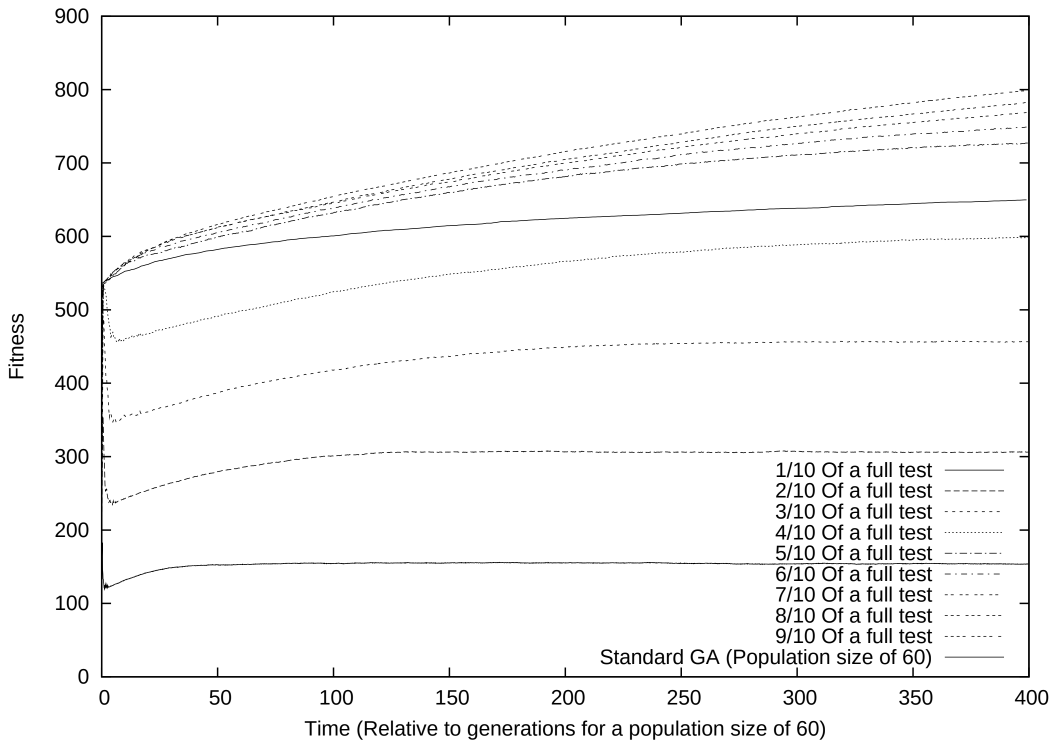

Figure C.8: Results of using a partial fitness function on the One Max Problem, replacing $\frac{8}{10}$ the population after each partial fitness test

The graph in Figure C.8 shows that when replacing $\frac{8}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, performs slightly better than for the same fitness functions when replacing $\frac{7}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

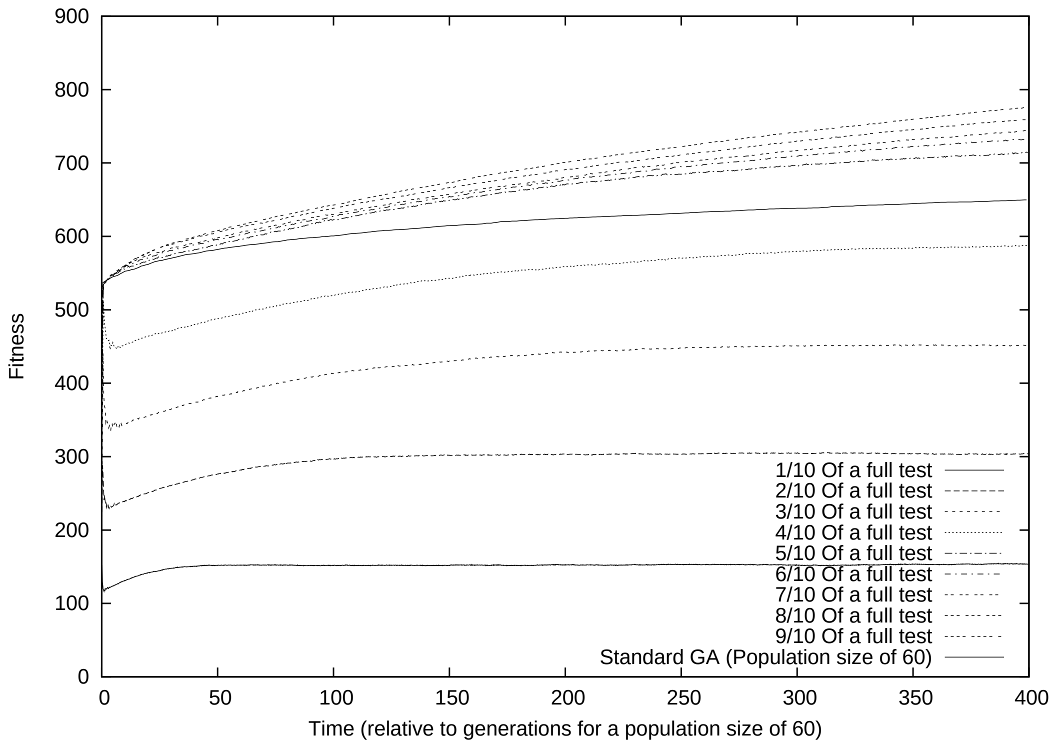

Figure C.9: Results of using a partial fitness function on the One Max Problem, replacing $\frac{9}{10}$ the population after each partial fitness test

The graph in Figure C.9 shows that when replacing $\frac{9}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{5}{10}$ to $\frac{9}{10}$ of an individual each generation, performs slightly worse than for the same fitness functions when replacing $\frac{8}{10}$ of the population. The partial fitness functions that perform $\frac{1}{10}$ to $\frac{4}{10}$ all performed worse than a standard GA tackling the same problem.

C.2 Deceptive Trap Function

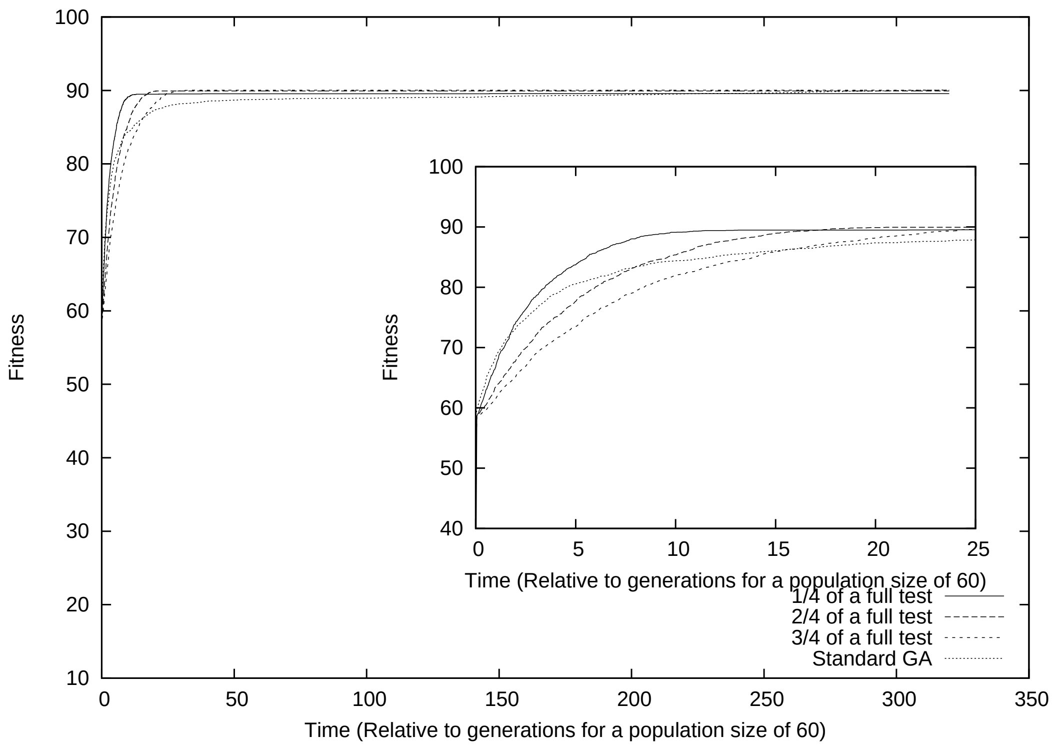

Figure C.10: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{1}{10}$ the population after each partial fitness tests

The graph in Figure C.10 shows that when replacing $\frac{1}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{1}{4}$ to $\frac{3}{4}$ of an individual are better than the fitness level produced by a standard GA on the same problem.

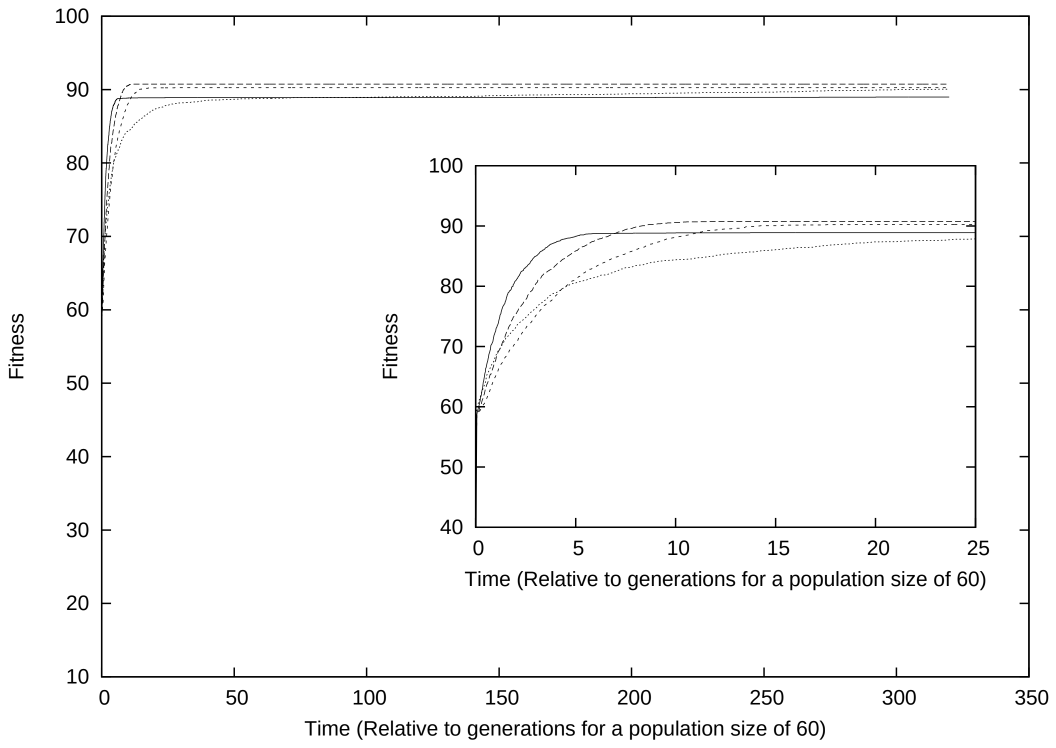

Figure C.11: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{2}{10}$ the population after each partial fitness tests

The graph in Figure C.11 shows that when replacing $\frac{2}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ of an individual each generation is better than when replacing only $\frac{1}{10}$ of the population. The partial fitness function testing $\frac{1}{4}$ of an individual each generation is about the same in the previous graph.

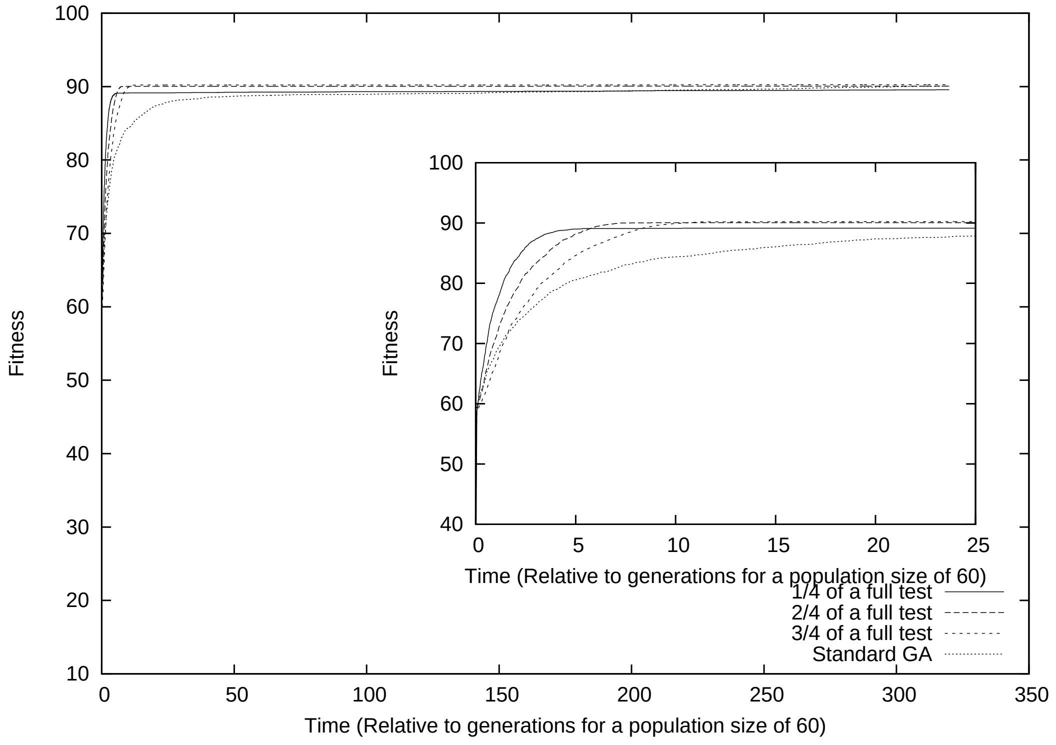

Figure C.12: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{3}{10}$ the population after each partial fitness tests

The graph in Figure C.12 shows that when replacing $\frac{3}{10}$ of the population after each generation the fitness level produced by the partial fitness functions testing $\frac{1}{4}$ and $\frac{3}{4}$ performs very similar to the previous graph. The partial fitness function testing $\frac{2}{4}$ of an individual each generation performs slightly worse than in the previous graph.

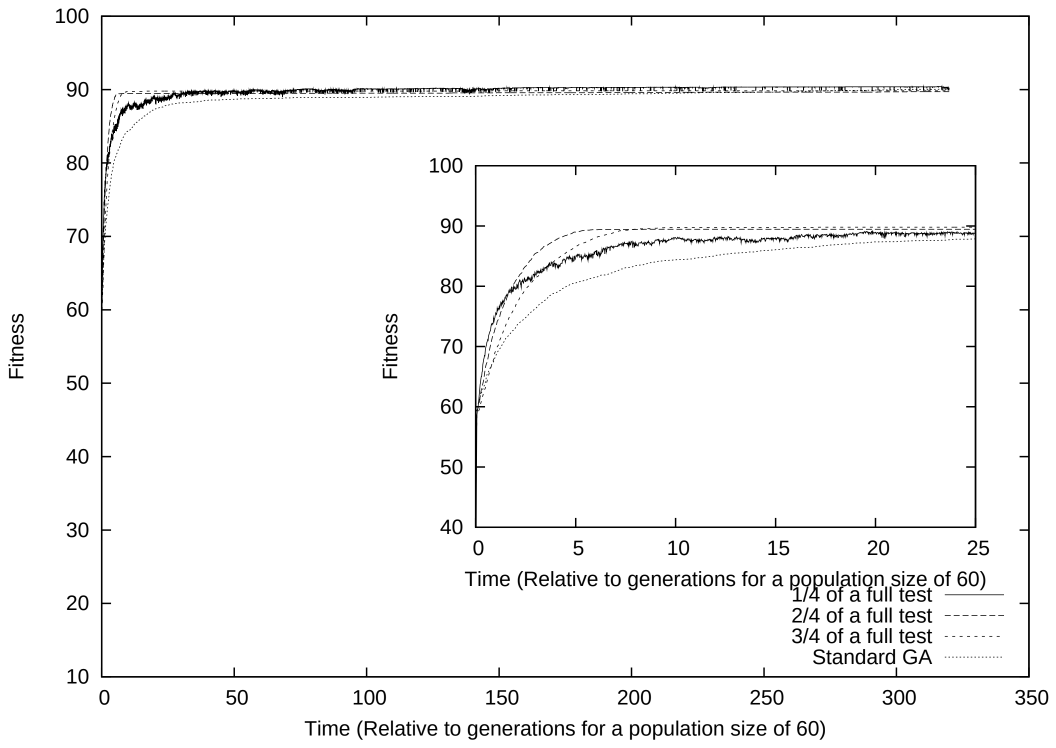

Figure C.13: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{4}{10}$ the population after each partial fitness tests

The graph in Figure C.13 shows that when replacing $\frac{4}{10}$ of the population after each generation the fitness level produced by the partial fitness function testing $\frac{1}{4}$ of an individual has not only dropped slightly but has also become unstable. The partial fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ produce very similar results to the previous graph.

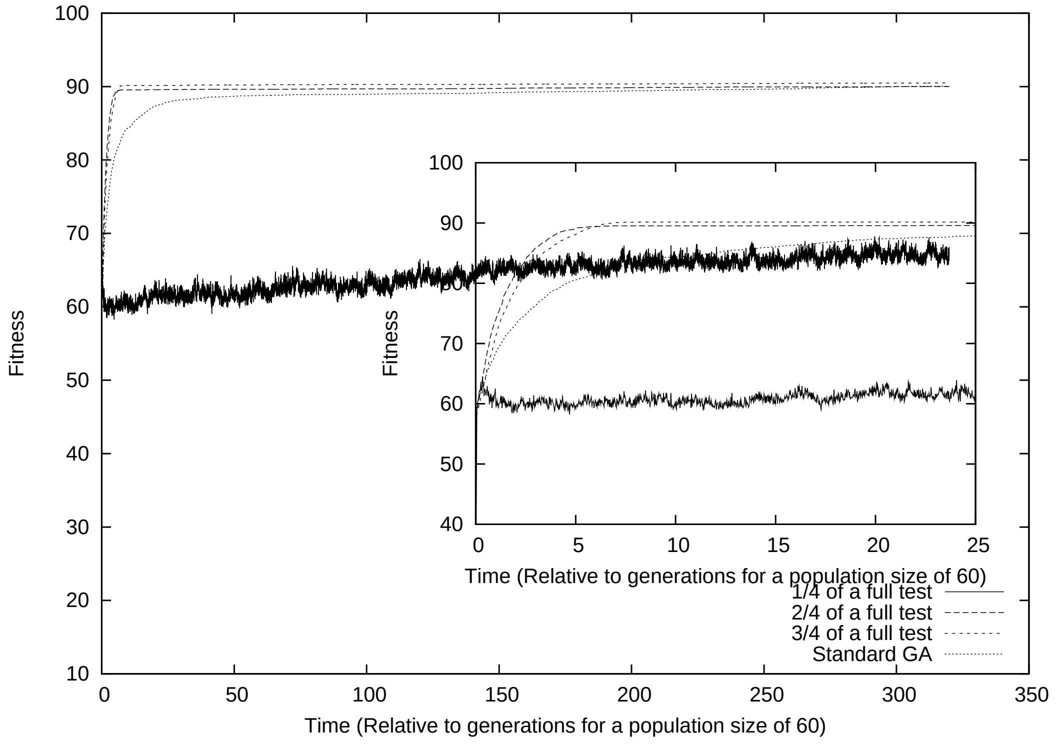

Figure C.14: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{5}{10}$ the population after each partial fitness tests

The graph in Figure C.14 shows that when replacing $\frac{5}{10}$ of the population after each generation the fitness level produced by the partial fitness function testing $\frac{1}{4}$ of an individual each generation has dropped well below the level achieved by the standard GA and has become even more unstable. The fitness levels achieved by the partial fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ are very similar to the previous graph.

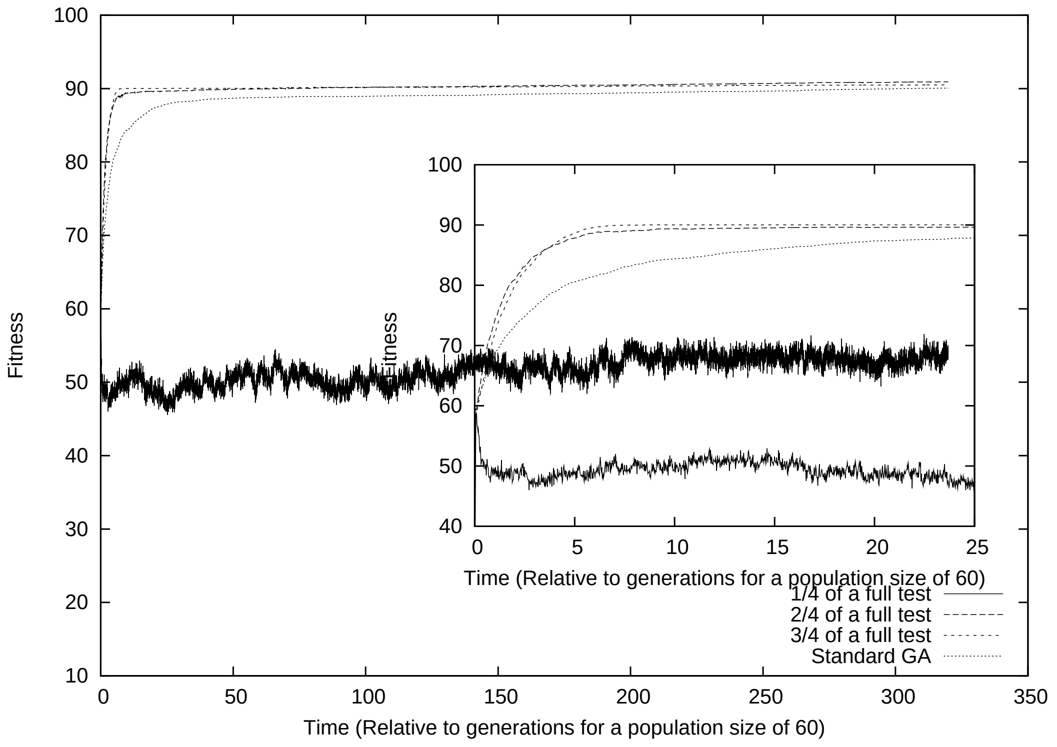

Figure C.15: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{6}{10}$ the population after each partial fitness tests

The graph in Figure C.15 shows that when replacing $\frac{6}{10}$ of the population after each generation the fitness levels produced by the partial fitness function testing $\frac{1}{4}$ of an individual each generation has dropped even lower than in the previous graph. The fitness functions testing $\frac{2}{4}$ and $\frac{3}{4}$ perform very similar to the previous graph.

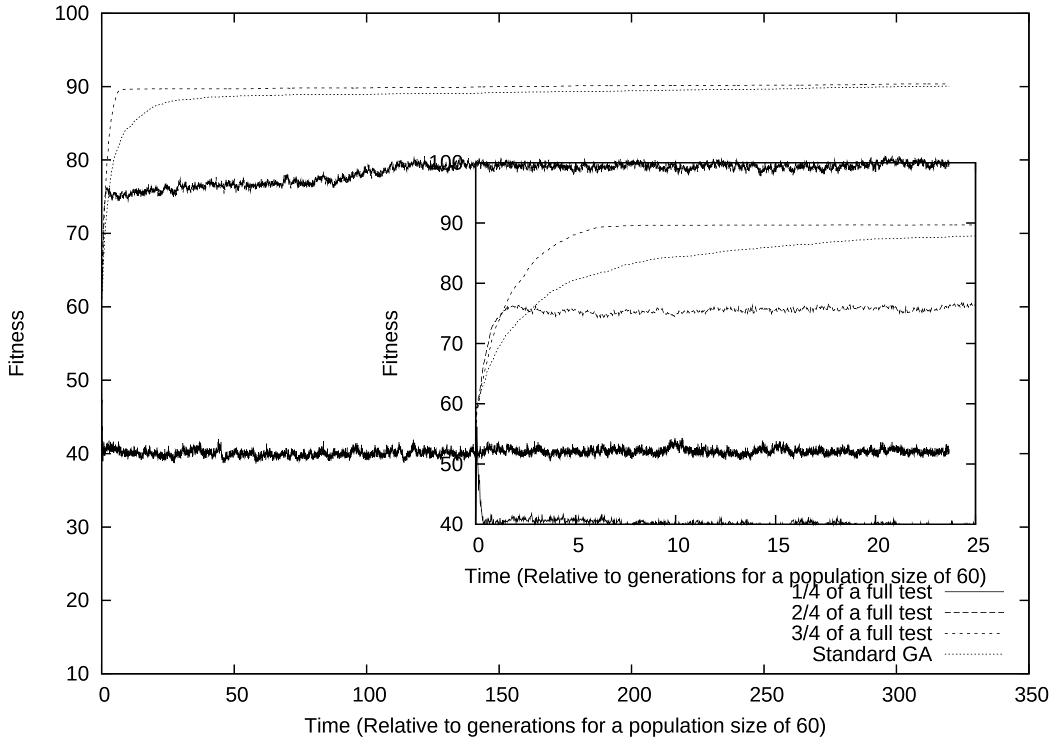

Figure C.16: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{7}{10}$ the population after each partial fitness tests

The graph in Figure C.16 shows that when replacing $\frac{7}{10}$ of the population after each generation the fitness levels produced by the partial fitness function testing $\frac{1}{4}$ of an individual each generation has again dropped even lower than in the previous graph. The fitness levels produced by the partial fitness function testing $\frac{2}{4}$ of an individual each generation has now dropped below those achieved by the standard GA and has also become unstable. The fitness levels achieved by the partial fitness function testing $\frac{3}{4}$ of an individual each generation are very similar to the previous graph.

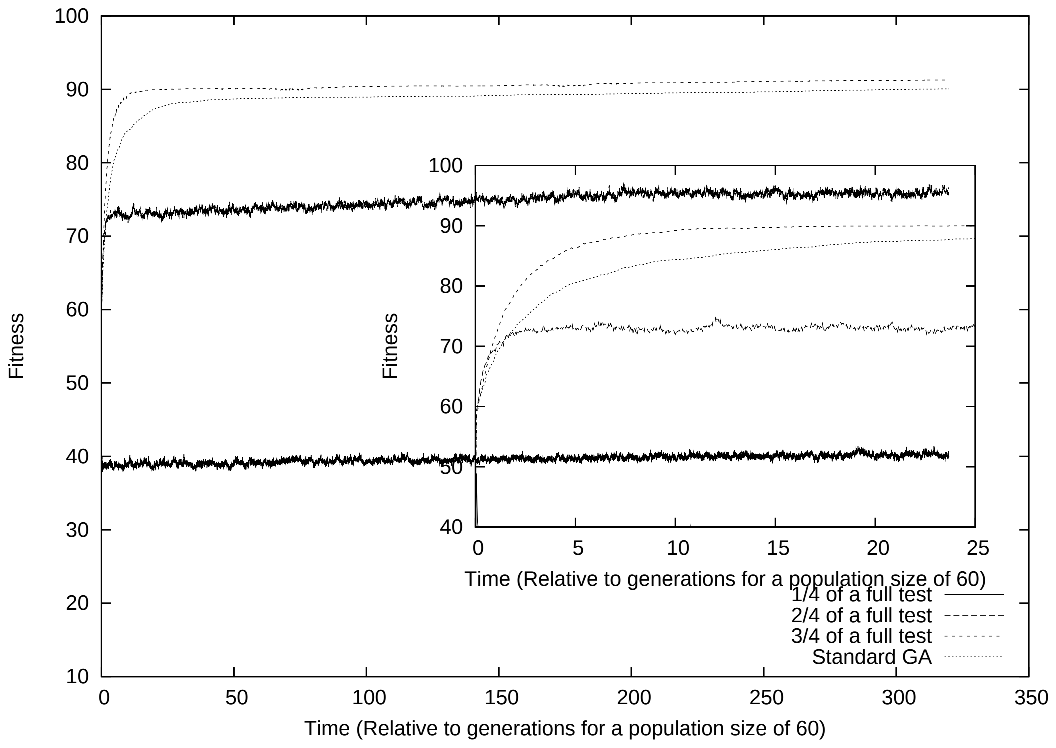

Figure C.17: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{8}{10}$ the population after each partial fitness tests

The graph in Figure C.17 shows that when replacing $\frac{8}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{1}{4}$ and $\frac{2}{4}$ are slightly worse than in the previous graph. The fitness levels of the partial fitness function $\frac{3}{4}$ is very similar still to the previous graph.

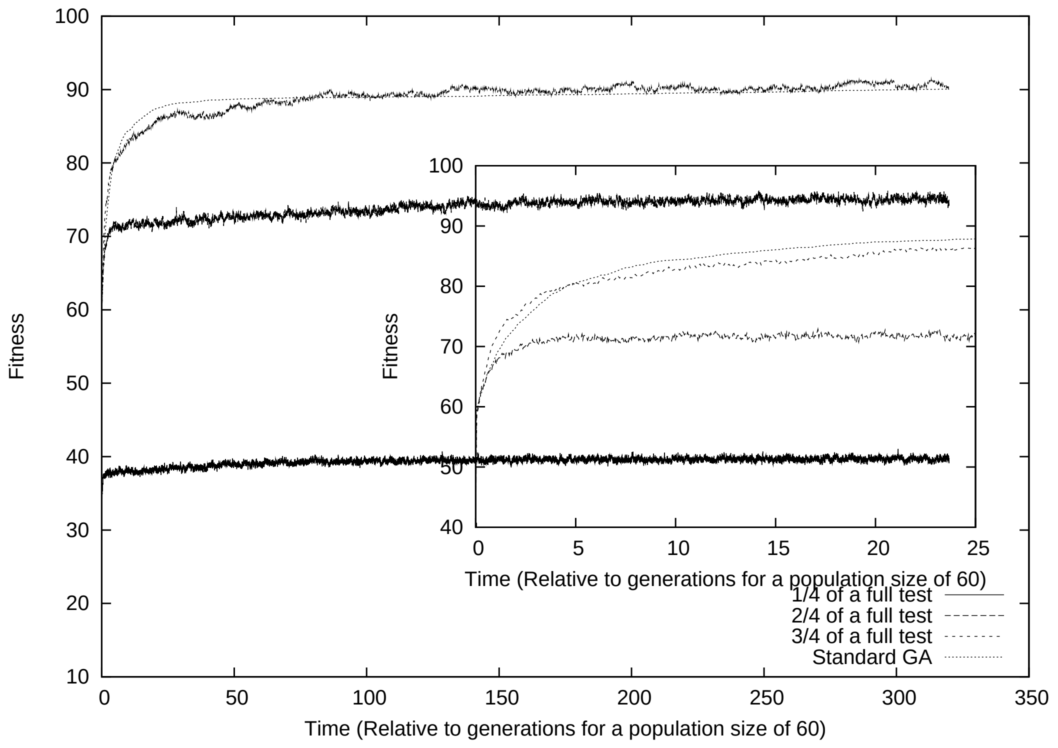

Figure C.18: Results of using a partial fitness function on the Deceptive Trap Function, replacing $\frac{9}{10}$ the population after each partial fitness tests

The graph in Figure C.18 shows that when replacing $\frac{9}{10}$ of the population after each generation the fitness levels produced by the partial fitness functions testing $\frac{1}{4}$ and $\frac{2}{4}$ are slightly worse than in the previous graph. The fitness levels of the partial fitness function $\frac{3}{4}$ has also dropped and has become unstable.

Appendix D

Plain Text and Cipher Text

The following two sections contain the cipher text and originating plain text used in chapter 6.

D.1 Original Plain Text

```
THEWA FERTU MBLER LOCKT HEWAF ERTUM BLERL OCKWA SDEVE LOPED
ASALO WCOST LOCKT HATOF FERED AREAS ONABL EDEGR EEOFS ECURI
TYTOT HEOWN ERTHE SELOC KSARE MAKEU POVER ONEFO URTHO FALLT
HELOC KSINT HEWOR LDTHE OUTSI DEOFT HELOC KRESE MBLES THEPI
NTUMB LERLO CKYET TOBED ISCUS SEDBU TUSES AMUCH SIMPL ERMEC
HANIS MWAFE RKEYW AYSUS UALLY HAVES IMPLE SIDEW ARDIN DENTI
ONSTH EKEYI SUSUA LLYSH ORTER THANT HATOF OTHER LOCKS BUTEQ
UALLY BROAD ITMAY BECUT ONONE ORBOT HSIDE SATWO SIDED WAFER
LOCKI SOFTE NCALL EDADO UBLEW AFERT HELOC KCONS ISTSO FFOUR
```

```
MAINP ARTST HEPLU GHOUS INGWH ICHCO NTAIN STHEW AFERS ANDSP

RINGS THESH ELLTH ECAML OCKIN GBOLT ANDTH ERETA INERT HEWAF

ERSAR ESOME TIMES REFER REDTO ASDIS CSBEC AUSET HEIRT OPAND

BOTTO MARER OUNDE DTOFI TINTO THECY LINDE RHERE ISADI AGRAM

KEYWA YPLUG DETAI LOFAW AFERT UMBLE RCUTA WAYSI DEVIE WSPRI

NGOFA WAFER LOCKK EYSLO TSPRI NGWIN GSPAC INGSC AMOPE RATES

THEBO LTRET AINER REARP LUGTH ESHEL LBODY OFTHE LOCKE ACHLO

CKHAS ASERI ESOFC HAMBE RSINW HICHT HEWAF ERSRE STTHE SESPA

CINGC LOSES TTOTH EFRON TOFTH ELOCK ISNUM BERED WITHO NEAND

THEIR NUMBE RSINC REASE TOWAR DTHEB ACKOF THELO CKPIC TUREA

NUMBE ROFTH EWAFE RSPLA CEDFA CETOF ACEIN THEPL UGSSP ACING

CHAMB ERSEA CHWAF ERISE QUALI NOVER ALLSI ZEBUT THEKE YSLOT

SAREO FVARY INGHE IGHTA METAL SPRIN GEXER TSPRE SSURE ONTHE

SPRIN GWING OFEAC HWAFE RFORC INGIT SLOWE RPART INTOT HESHE

LLSLO CKING GROOV ESWHI CHLET STHEL OWERP ORTIO NHANG ABOUT

MIDWA YINTO THEKE YWAYL OOKIN GINTO THELO CKYOU SHOUL DBEAB

LETOS EETHI STHES EWAFE RSACT TOHOL DTHEP LUGAN DSHEL LTOGE

THERP REVEN TINGT HELOC KFROM TURNI NGWHE NTHEC ORREC TKEYI

SINSE RTEDI TGOES THROU GHTHE KEYSL OTSON EACHW AFERR AISIN

GTHEW AFERS OUTOF THELO CKING GROOV ETHEK EYMUS THAVE THEAP

PROPR IATED EPTHO FCUTI NEACH POSIT IONTO RAISE THEWA FERTH

ECORR ECTAM OUNTT HEDEP THOFT HEKEY SCUTA NDTHE LENGT HOFTH

EWAFE RSKEY SLOTI SANYO NEOFF IVEDI FFERE NTDEP THSTH ESHOR

TERTH ETOPE DGEOF THEWA FERSK EYSLO TTHEL OWERT HEKEY CUTDE

PTHVA LUEFO RINST ANCET HENUM BERSL OTTHE SLOTT HATIS THELA
```

```
RGEST WOULD REQUI RETHE SHALL OWEST CUTIN THEKE YNORM ALLYL

OCKMA NUFAC TURER SPLAC EANUM BERFO URORF IVEWA FERNE ARTHE

KEYHO LETOB LOCKT HEVIE WOFTH EBACK WAFER SALSO NOTET HATTH

ESAME TYPEO FWAFE RMAYA PPEAR SEVER ALTIM ESINT HESAM ELOCK

ABOVE SOMEB RANDS OFWAF ERTUM BLERL OCKYO UWILL SEEAS MALLH

OLEWH ENTHE LOCKH ASBEE NUNLO CKEDY OUCAN REMOV ETHEE NTIRE

LOCKP LUGBY INSER TINGA PIECE OFSTI FFWIR EINTO THISH OLEAN

DDEPR ESSIN GTHER ETAIN ERTHO UGHNO WHERE NEARA SSECU REAST

HEPIN TUMBL ERLOC KTHEW AFERT UMBLE RISAV ERYPO PULAR LOWCO

STLOC KTHEL OCKIS NORMA LLYFO UNDON CHEAP ERCAB INETS ANDDE

SKSSO MEPAD LOCKS SOMEA UTOMO BILEL OCKSL OCKIN GHAND LESAN

DTRAI LERDO ORSWH EREMO RESEC URITY ISDES IREDT HEDOU BLEWA

FERTY PEISU SEDPR OVIDI NGWAF ERSON THETO PANDB OTTOM OFTHE

KEYWA YPICK INGTH OUGHH ARDER TOPIC KTHEN THEWA RDEDL OCKTH

EWAFE RLOCK ISSTI LLEAS YTOCI RCUMV ENTTH ISISA NEXCE LLENT

LOCKT OPRAC TICEO NBECA USETH ETECH NIQUE SREQU IREDT OPICK

ITARE APPLI CABLE TOTHE PINTU MBLER LOCKA SWELL LIKET HELEV

ERLOC KPICK INGTH EWAFE RTUMB LERLO CKREQ UIRES USEOF ATENS

IONWR ENCHA NDAPI CKAVA RIETY OFTHE DIFFE RENTP ICKSC ANBEU

SEDIN CLUDI NGTHE RAKET HEHOO KTHEH ALFDI AMOND ANDTH EHALF

ROUND PICKS ELECT IONDE PENDS ONTHE SIZEO FTHEL OCKTH EDIST

ANCEB ETWEE NEACH WAFER ANDPE RSONA LPREF ERENC ERAKI NGONE

OFTHE MOSTC OMMON METHO DSOFP ICKIN GTHEW AFERT UMBLE RLOCK

ISBYR AKING TORAK ETHEL OCKIN SERTT HETEN SIONW RENCH ISINS

ERTED JUSTI NSIDE THEKE YWAYS TOPPI NGSHO RTOFT HEFIR STWAF
```

```
ERAND FLUSH WITHT HEBOT TOMOF THEKE YWAYA PPLYM ODERA TETEN

SIONT OTHEW RENCH IFYOU APPLY TOOMU CHTEN SIONT HEWAF ERSWI

LLBIN DANDN OTBEA BLETO MOVEI NTOAL IGNME NTONC EYOUH AVETH

ETENS IONWR ENCHI NPLAC EINSE RTEIT HERTH ERAKE ORHAL FROUN

DPICK INTOT HEKEY WAYDO NTWOR RYABO UTFEE LINGT HETUM BLERS

INSTE ADCON CENTR ATEON APPLY INGUN IFORM PRESS URETO THEMA

SYOUM OVETH ERAKE INAND OUTOF THEKE YWAYI NASCR UBBIN GMOTI

ONTHI SSCRU BBING MOTIO NSHOU LDCAU SETHE WAFER STOLI FTINT

OALIG NMENT ASTHE YARET HROWN UPAND DOWNI NTHEI RSPAC INGST

HISME THODI SUSUA LLYQU ITEEF FECTI VEONM OSTWA FERLO CKSAN

DSHOU LDALW AYSBE TRIED FIRST MANIP ULATI NGIND IVIDU ALWAF

ERSIF THELO CKDOE SNOTR ESPON DTORA KINGY OUCAN TRYUS INGTH

EHALF DIAMO NDPIC KTOEA CHWAF ERINT OALIG NMENT ONEBY ONEWH

ILEMA INTAI NINGL IGHTB UTCON SISTE NTPRE SSURE WITHT HETEN

SIONW RENCH USETH EPICK TOLIF TEACH WAFER INTOA LIGNM ENTAT

THESH EARLI NESTA RTING FROMT HEBAC KMOST TUMBL ERONC EITRE

ACHES THEPR OPERA LIGNM ENTYO USHOU LDFEE LORHE ARASL IGHTC

LICKA NDTHE PLUGW ILLTU RNEVE RSOSL IGHTL YRELI EVING ABITO

FPRES SUREO NTHEW RENCH CONTI NUEON EBYON EWORK INGOU TWARD

UNTIL EACHT UMBLE RHASB EENAL IGNED ANDTH ELOCK OPENS VIBRA

TIONP ICKIN GOFTE NYOUC ANUSE ATECH NIQUE CALLE DVIBR ATION

PICKI NGTOO PENAW AFERT UMBLE RLOCK THISU SESAT OOLKN OWNAS

ASNAP PERPI CKORA LOCKP ICKGU NTHES EARED ESCRI BEDIN THELO

CKPIC KINGT OOLSS ECTIO NOFTH ISART ICLET OUSET HESNA PPERP

ICKMA INTAI NALIG HTTEN SIONW ITHTH EWREN CHAND INSER TTHET
```

```
IPOFT HEPIC KINTO THEKE YWAYJ USTTO UCHIN GTHEB OTTOM OFTHE
TUMBL ERSTH ENUSE THETH UMBWH ICHRE STSAL ONGTH ETOPE DGEOF
THEPI CKTOD EPRES STHET OPLOO PLETT HETHU MBSLI DEOFF THECO
MPRES SEDPA RTOFT HEPIC KPERM ITTIN GITTO SNAPB ACKIT WILLT
HENST RIKEA LIGHT BLOWT OTHET UMBLE RSPOP PINGT HEMUP UNTIL
THEYA REHEL DINPL ACEAT THESH EARLI NEREP EATED SNAPS WHILE
MAINT AININ GTENS IONWI THTHE WRENC HUSUA LLYRE SULTS INALI
GNING ALLTH ETUMB LERSA NDTHU SOPEN INGTH ELOCK THELO CKPIC
KGUNW ORKSA UTOMA TICAL LYWIT HATRI GGERD EVICE THATS NAPSI
TSWIR EPICK UPINT HEKEY WAYPI CKING DOUBL EWAFE RLOCK SDOUB
LEWAF ERLOC KSARE PICKE DTHES AMEWA YASSI NGLEW AFERL OCKSB
UTTHE RETWO SIDES TOTHE STORY NOTON LYMUS TYOUA LIGNA LLTHE
TOPWA FERSB UTTHE BOTTO MONES ASWEL LYOUC ANPUR CHASE SPECI
ALDES IGNED TENSI ONWRE NCHES WITHW ILLLE TYOUT HENUS EABAL
LPICK TOPIC KBOTH SETSO FWAFE RSALT ERNAT IVELY YOUCA NUSEA
STAND ARDTE NSION WRENC HINTH ECENT EROFT HEKEY WAYUS INGAH
ALFDI AMOND PICKO NCEYO UHAVE PICKE DONES ETSIM PLYRE VERSE
THEPI CKAND PICKT HEOTH ERITM AYTAK EAFEW TRIES BEFOR EYOUA
REABL ETOHO LDALL THEWA FERSI NPLAC E
```

D.2 Cipher Text

```
GSVDZ UVIGF NYOVI OLXPG SVDZU VIGFN YOVIO LXPDZ HWVEV OLKVW
ZHZOL DXLHG OLXPG SZGLU UVIVW ZIVZH LMZYO VWVTI VVLUH VXFIR
GBGLG SVLDM VIGSV HVOLX PHZIV NZPVF KLEVI LMVUL FIGSL UZOOG
```

```
SVOLX PHRMG SVDLI OWGSV LFGHR WVLUG SVOLX PIVHV NYOVH GSVKR

MGFNY OVIOL XPBVG GLYVW RHXFH HVWYF GFHVH ZNFXS HRNKO VINVX

SZMRH NDZUV IPVBD ZBHFH FZOOB SZEVH RNKOV HRWVD ZIWRM WVMGR

LMHGS VPVBR HFHFZ OOBHS LIGVI GSZMG SZGLU LGSVI OLXPH YFGVJ

FZOOB YILZW RGNZB YVXFG LMLMV LIYLG SHRWV HZGDL HRWVW DZUVI

OLXPR HLUGV MXZOO VWZWL FYOVD ZUVIG SVOLX PXLMH RHGHL UULFI

NZRMK ZIGHG SVKOF TSLFH RMTDS RXSXL MGZRM HGSVD ZUVIH ZMWHK

IRMTH GSVHS VOOGS VXZNO LXPRM TYLOG ZMWGS VIVGZ RMVIG SVDZU

VIHZI VHLNV GRNVH IVUVI IVWGL ZHWRH XHYVX ZFHVG SVRIG LKZMW

YLGGL NZIVI LFMWV WGLUR GRMGL GSVXB ORMWV ISVIV RHZWR ZTIZN

PVBDZ BKOFT WVGZR OLUZD ZUVIG FNYOV IXFGZ DZBHR WVERV DHKIR

MTLUZ DZUVI OLXPP VBHOL GHKIR MTDRM THKZX RMTHX ZNLKV IZGVH

GSVYL OGIVG ZRMVI IVZIK OFTGS VHSVO OYLWB LUGSV OLXPV ZXSOL

XPSZH ZHVIR VHLUX SZNYV IHRMD SRXSG SVDZU VIHIV HGGSV HVHKZ

XRMTX OLHVH GGLGS VUILM GLUGS VOLXP RHMFN YVIVW DRGSL MVZMW

GSVRI MFNYV IHRMX IVZHV GLDZI WGSVY ZXPLU GSVOL XPKRX GFIVZ

MFNYV ILUGS VDZUV IHKOZ XVWUZ XVGLU ZXVRM GSVKO FTHHK ZXRMT

XSZNY VIHVZ XSDZU VIRHV JFZOR MLEVI ZOOHR AVYFG GSVPV BHOLG

HZIVL UEZIB RMTSV RTSGZ NVGZO HKIRM TVCVI GHKIV HHFIV LMGSV

HKIRM TDRMT LUVZX SDZUV IULIX RMTRG HOLDV IKZIG RMGLG SVHSV

OOHOL XPRMT TILLE VHDSR XSOVG HGSVO LDVIK LIGRL MSZMT ZYLFG

NRWDZ BRMGL GSVPV BDZBO LLPRM TRMGL GSVOL XPBLF HSLFO WYVZY

OVGLH VVGSR HGSVH VDZUV IHZXG GLSLO WGSVK OFTZM WHSVO OGLTV

GSVIK IVEVM GRMTG SVOLX PUILN GFIMR MTDSV MGSVX LIIVX GPVBR

HRMHV IGVWR GTLVH GSILF TSGSV PVBHO LGHLM VZXSD ZUVII ZRHRM
```

```
TGSVD ZUVIH LFGLU GSVOL XPRMT TILLE VGSVP VBNFH GSZEV GSVZK

KILKI RZGVW VKGSL UXFGR MVZXS KLHRG RLMGL IZRHV GSVDZ UVIGS

VXLII VXGZN LFMGG SVWVK GSLUG SVPVB HXFGZ MWGSV OVMTG SLUGS

VDZUV IHPVB HOLGR HZMBL MVLUU REVWR UUVIV MGWVK GSHGS VHSLI

GVIGS VGLKV WTVLU GSVDZ UVIHP VBHOL GGSVO LDVIG SVPVB XFGWV

KGSEZ OFVUL IRMHG ZMXVG SVMFN YVIHO LGGSV HOLGG SZGRH GSVOZ

ITVHG DLFOW IVJFR IVGSV HSZOO LDVHG XFGRM GSVPV BMLIN ZOOBO

LXPNZ MFUZX GFIVI HKOZX VZMFN YVIUL FILIU REVDZ UVIMV ZIGSV

PVBSL OVGLY OLXPG SVERV DLUGS VYZXP DZUVI HZOHL MLGVG SZGGS

VHZNV GBKVL UDZUV INZBZ KKVZI HVEVI ZOGRN VHRMG SVHZN VOLXP

ZYLEV HLNVY IZMWH LUDZU VIGFN YOVIO LXPBL FDROO HVVZH NZOOS

LOVDS VMGSV OLXPS ZHYVV MFMOL XPVWB LFXZM IVNLE VGSVV MGRIV

OLXPK OFTYB RMHVI GRMTZ KRVXV LUHGR UUDRI VRMGL GSRHS LOVZM

WWVKI VHHRM TGSVI VGZRM VIGSL FTSML DSVIV MVZIZ HHVXF IVZHG

SVKRM GFNYO VIOLX PGSVD ZUVIG FNYOV IRHZE VIBKL KFOZI OLDXL

HGOLX PGSVO LXPRH MLINZ OOBUL FMWLM XSVZK VIXZY RMVGH ZMWWV

HPHHL NVKZW OLXPH HLNVZ FGLNL YROVO LXPHO LXPRM TSZMW OVHZM

WGIZR OVIWL LIHDS VIVNL IVHVX FIRGB RHWVH RIVWG SVWLF YOVDZ

UVIGB KVRHF HVWKI LERWR MTDZU VIHLM GSVGL KZMWY LGGLN LUGSV

PVBDZ BKRXP RMTGS LFTSS ZIWVI GLKRX PGSVM GSVDZ IWVWO LXPGS

VDZUV IOLXP RHHGR OOVZH BGLXR IXFNE VMGGS RHRHZ MVCXV OOVMG

OLXPG LKIZX GRXVL MYVXZ FHVGS VGVXS MRJFV HIVJF RIVWG LKRXP

RGZIV ZKKOR XZYOV GLGSV KRMGF NYOVI OLXPZ HDVOO ORPVG SVOVE

VIOLX PKRXP RMTGS VDZUV IGFNY OVIOL XPIVJ FRIVH FHVLU ZGVMH

RLMDI VMXSZ MWZKR XPZEZ IRVGB LUGSV WRUUV IVMGK RXPHX ZMYVF
```

```
HVWRM XOFWR MTGSV IZPVG SVSLL PGSVS ZOUWR ZNLMW ZMWGS VSZOU

ILFMW KRXPH VOVXG RLMWV KVMWH LMGSV HRAVL UGSVO LXPGS VWRHG

ZMXVY VGDVV MVZXS DZUVI ZMWKV IHLMZ OKIVU VIVMX VIZPR MTLMV

LUGSV NLHGX LNNLM NVGSL WHLUK RXPRM TGSVD ZUVIG FNYOV IOLXP

RHYBI ZPRMT GLIZP VGSVO LXPRM HVIGG SVGVM HRLMD IVMXS RHRMH

VIGVW QFHGR MHRWV GSVPV BDZBH GLKKR MTHSL IGLUG SVURI HGDZU

VIZMW UOFHS DRGSG SVYLG GLNLU GSVPV BDZBZ KKOBN LWVIZ GVGVM

HRLMG LGSVD IVMXS RUBLF ZKKOB GLLNF XSGVM HRLMG SVDZU VIHDR

OOYRM WZMWM LGYVZ YOVGL NLEVR MGLZO RTMNV MGLMX VBLFS ZEVGS

VGVMH RLMDI VMXSR MKOZX VRMHV IGVRG SVIGS VIZPV LISZO UILFM

WKRXP RMGLG SVPVB DZBWL MGDLI IBZYL FGUVV ORMTG SVGFN YOVIH

RMHGV ZWXLM XVMGI ZGVLM ZKKOB RMTFM RULIN KIVHH FIVGL GSVNZ

HBLFN LEVGS VIZPV RMZMW LFGLU GSVPV BDZBR MZHXI FYYRM TNLGR

LMGSR HHXIF YYRMT NLGRL MHSLF OWXZF HVGSV DZUVI HGLOR UGRMG

LZORT MNVMG ZHGSV BZIVG SILDM FKZMW WLDMR MGSVR IHKZX RMTHG

SRHNV GSLWR HFHFZ OOBJF RGVVU UVXGR EVLMN LHGDZ UVIOL XPHZM

WHSLF OWZOD ZBHYV GIRVW URIHG NZMRK FOZGR MTRMW RERWF ZODZU

VIHRU GSVOL XPWLV HMLGI VHKLM WGLIZ PRMTB LFXZM GIBFH RMTGS

VSZOU WRZNL MWKRX PGLVZ XSDZU VIRMG LZORT MNVMG LMVYB LMVDS

ROVNZ RMGZR MRMTO RTSGY FGXLM HRHGV MGKIV HHFIV DRGSG SVGVM

HRLMD IVMXS FHVGS VKRXP GLORU GVZXS DZUVI RMGLZ ORTMN VMGZG

GSVHS VZIOR MVHGZ IGRMT UILNG SVYZX PNLHG GFNYO VILMX VRGIV

ZXSVH GSVKI LKVIZ ORTMN VMGBL FHSLF OWUVV OLISV ZIZHO RTSGX

ORXPZ MWGSV KOFTD ROOGF IMVEV IHLHO RTSGO BIVOR VERMT ZYRGL

UKIVH HFIVL MGSVD IVMXS XLMGR MFVLM VYBLM VDLIP RMTLF GDZIW
```

```
FMGRO VZXSG FNYOV ISZHY VVMZO RTMVW ZMWGS VOLXP LKVMH ERYIZ

GRLMK RXPRM TLUGV MBLFX ZMFHV ZGVXS MRJFV XZOOV WERYI ZGRLM

KRXPR MTGLL KVMZD ZUVIG FNYOV IOLXP GSRHF HVHZG LLOPM LDMZH

ZHMZK KVIKR XPLIZ OLXPK RXPTF MGSVH VZIVW VHXIR YVWRM GSVOL

XPKRX PRMTG LLOHH VXGRL MLUGS RHZIG RXOVG LFHVG SVHMZ KKVIK

RXPNZ RMGZR MZORT SGGVM HRLMD RGSGS VDIVM XSZMW RMHVI GGSVG

RKLUG SVKRX PRMGL GSVPV BDZBQ FHGGL FXSRM TGSVY LGGLN LUGSV

GFNYO VIHGS VMFHV GSVGS FNYDS RXSIV HGHZO LMTGS VGLKV WTVLU

GSVKR XPGLW VKIVH HGSVG LKOLL KOVGG SVGSF NYHOR WVLUU GSVXL

NKIVH HVWKZ IGLUG SVKRX PKVIN RGGRM TRGGL HMZKY ZXPRG DROOG

SVMHG IRPVZ ORTSG YOLDG LGSVG FNYOV IHKLK KRMTG SVNFK FMGRO

GSVBZ IVSVO WRMKO ZXVZG GSVHS VZIOR MVIVK VZGVW HMZKH DSROV

NZRMG ZRMRM TGVMH RLMDR GSGSV DIVMX SFHFZ OOBIV HFOGH RMZOR

TMRMT ZOOGS VGFNY OVIHZ MWGSF HLKVM RMTGS VOLXP GSVOL XPKRX

PTFMD LIPHZ FGLNZ GRXZO OBDRG SZGIR TTVIW VERXV GSZGH MZKHR

GHDRI VKRXP FKRMG SVPVB DZBKR XPRMT WLFYO VDZUV IOLXP HWLFY

OVDZU VIOLX PHZIV KRXPV WGSVH ZNVDZ BZHHR MTOVD ZUVIO LXPHY

FGGSV IVGDL HRWVH GLGSV HGLIB MLGLM OBNFH GBLFZ ORTMZ OOGSV

GLKDZ UVIHY FGGSV YLGGL NLMVH ZHDVO OBLFX ZMKFI XSZHV HKVXR

ZOWVH RTMVW GVMHR LMDIV MXSVH DRGSD ROOOV GBLFG SVMFH VZYZO

OKRXP GLKRX PYLGS HVGHL UDZUV IHZOG VIMZG REVOB BLFXZ MFHVZ

HGZMW ZIWGV MHRLM DIVMX SRMGS VXVMG VILUG SVPVB DZBFH RMTZS

ZOUWR ZNLMW KRXPL MXVBL FSZEV KRXPV WLMVH VGHRN KOBIV EVIHV

GSVKR XPZMW KRXPG SVLGS VIRGN ZBGZP VZUVD GIRVH YVULI VBLFZ

IVZYO VGLSL OWZOO GSVDZ UVIHR MKOZX V
```