Partner Selection in Evolutionary Computation

NUI Galway
OÉ Gaillimh

Sean Harkin

13344471
BE Computer and Electronic Engineering
Project Supervisor Dr Colm O'Riordan

Abstract

The aim of this project is to explore the performance of partner selection algorithms in the field of evolutionary computation, with a particular focus on their use in Genetic algorithms. Several methods will be tested for their ability to perform on a range of problem landscapes of varying difficulty, this work will also explore the effect of modifying partner selection methods and their impact on performance .

Acknowledgements

I would like to thank my supervisor, Dr Colm O'Riordan, for his patience and guidance throughout the course of this project his knowledge and expertise were indispensable in the creating and completion of this project. I would also like to thank Dr Seamus Hill and Jane Holland for their advice and direction over the past few months. Finally I would like to thank family and friends for their continued support over the past number of years.

Table of Contents

Introduction
- 1.1 Genetic Algorithms
- 1.2 Partner Selection in Genetic Algorithms
- 1.3 Scope and Methodology
- 1.4 Report Layout

Related Work
- 2.1 Evolutionary Computation
- 2.2 Genetic Algorithms
- 2.3 Fitness Function
- 2.4 Problem Landscapes
- 2.5 Crossover
- 2.6 Diversity
- 2.7 Partner Selection
  - 2.7.1 Non- overlapping generation models
  - 2.7.2 Overlapping-Generation Models
- 2.8 Epistasis

Design and Implementation
- 3.1 Population Creation
- 3.2 Problem Landscapes
- 3.3 Fitness Functions
  - 3.3.1 Fitness Sum
  - 3.3.2 Deceptive Detection
  - 3.3.3 Phenotypic Profile
- 3.4 Tournament selection
- 3.5 Partner Selection
  - 3.6.1 Fitness only
  - 3.6.2 Hamming
  - 3.6.3 Phenotypic matching
- 3.6 Crossover
  - 3.5.1 Single and Multi Point Crossover
  - 3.5.3 Weighted Crossover
  - 3.5.4 Uniform Crossover
- 3.7 Mutation

3.8 Alleles
3.9 Elitism
3.11 Results Gathering

4 Testing and Results
4.1 Experiment 1: Standard Landscapes
4.2 Experiment 2: Varying Allele
4.3 Experiment 3: Uniform crossover
4.4. Experiment 4: Epistasis

5 Conclusion
5.1 Future Work

Introduction

This section will provide a brief introduction to the field of evolutionary computation and genetic algorithms, as well as better, define the outline of the project.

1.1 Genetic Algorithms

Genetic Algorithms are a field within Evolutionary computation which originate from the computer modelling of Darwinian evolution observed in nature. These algorithms utilise the knowledge learnt from over a centuries research into the theory of evolution. Genetic algorithms (‘GA's) in particular pay close attention to the behaviour and properties of genes in evolution using this mechanism to develop and evolve solutions to a problem landscape. GA's can be used to solve a wide range of problems from a travelling salesman problem to more complex problems such as the design of antennas for spacecraft. In simple problem landscapes, the solution can be encoded as a string of 0's and 1's this can be altered if more complex problems are being attempted.

The first step in this process is to generate a random population of problem solutions these problem solutions will be used as the first generation for the GA, each individual from this
the population will be evaluated using some fitness function such as summing all of the ones in the solution. The fittest solutions are then chosen and used to produce offspring and create the next generation. Several methods exist for deciding which solutions should be paired to produce offspring, one of which is to select each one from the population then select four others from the same population and choose which one matches best with chosen solution and repeat this process until the next generation is created. Mutation is key to the evolutionary process without it organisms would not evolve and so in GA's, it is vital to use an effective mutation rate to ensure the advancement and improvement of the problem solutions in the population. The mutation occurs during the crossover and copying of genes in the creation of a new organism. Once all of these processes have been completed you will be left with a new population and the process starts again and repeats itself until a suitable solution has been found.

1.2 Partner Selection in Genetic Algorithms

In most species there is an imbalance in the cost of raising offspring in terms of time between the sexes, this disparity can lead to different approaches in finding a mate.Such as in lions where the majority of the investment for raising the young is the responsibility of the female, while the male's time investment is minimal this results in a great variety of methods employed by species to attract a good partner. One example is that of male birds of paradise which develop elaborate and colourful feathers which cost vast amounts of time and energy to develop but show a potential mate that they are capable of hunting or gathering plenty of food which is a desirable trait to have in your offspring, because to this the female birds of paradise can simply choose the most suitable male to reproduce with. These behaviours can

provide useful starting points in the development and testing of various partner selection algorithms. In GA's partner selection is an import factor contributing to effectiveness and quality of a GA's solutions. The partner selection process is used to ensure the fittest and most suitable solutions will be recombined the fittest solutions.There are already many methodologies developed to optimise the process of partner selection many of which will be explored and discussed in latter chapters each of which has its own strengths and weaknesses.

1.3 Scope and Methodology

The main aim of this project is to test the ability and performance of a number of partner selection methods in genetic algorithms and to gather and measure their performance on a number of different problem landscapes.In order to do this, I had to first build a GA the details of which are found in Chapter 3 Design and Implementation. Several metrics will be used to assess the ability of the algorithm on the problem, one of which is the number of generation taken to find an ideal solution for that problem referred to as generations taken in graphs.Measures of the generations fitness will also be used to gauge the performance of the algorithm these will often take the form of best average and worst fitness of a generation and help to decipher the rates and which the algorithm converges to the ideal. A fail rate measure will also be utilised to test the ability of the system.

1.4 Report Layout

Chapter 1 of the report will contain an introduction to the topics covered in a latter chapter of the report as well as outlining the approach taken in the project. Chapter 2 will be a comprehensive literature review on the topics of genetic algorithms and partner selection. Design and implementation are found in Chapter 3 this is a detailed report of the working of the GA. Chapter 4 contains the testing, results and analyses of the output of the GA.Finally, Chapter 5 will contain the conclusion on the results and the details of further work.

Related Work

The contents of this chapter will define and summarise the work undertaken in the field of evolutionary computation and genetic algorithms. It will focus on and pay close attention to features of genetic algorithms which are were used in the project, It will also help to put the work into context as it relates to the broader field of study.

2.1 Evolutionary Computation

Evolutionary computation(EV) is a group of bio-inspired algorithms used for global optimisation. All EV algorithm share common traits and methods for problem-solving all start with some sample population of solutions which can be randomly generated or pre-selected and groomed for the problem at hand. These sample solutions once created would then be evaluated using some performance measure which will determine their eligibility for progressing to successive generations, the algorithm would then remove the weak solutions from the population leaving the fit and strong solutions to proliferate. This method of removing the weak solutions and leaving the strong models the evolutionary concept of Darwinian natural selection.

Evolutionary computation is a relatively young field of study and has only been around since the mid-1950's[1]. Evidence of the use of computer modelling to better understand natural selection evolution appeared before that time but it was not until the 1950's that articles describing its use had been published. During these years the first steps were taken into this new field of study however it was not until the next decade that these concepts would get refined into the ones one know today. One of these branches was evolutionary programming which first appeared in 1967 proposed by Lawrence Fogal in San Diego, In Berlin, another branch was taking root that of evolutionary strategies first proposed by three students, Bienert, Rechenberg and Schwefel in 1965. Finally, genetic algorithms first appeared in 1967 in a paper titled Genetic Algorithms by John H Holland at the University of Michigan-Ann Arbor. One established these three branches grew independently of one and are still evolving even to the current day.

2.2 Genetic Algorithms

Genetic algorithms (GA) mimic the evolutionary principles set forth by Darwin in the 1800's they utilise the knowledge learnt from observing nature and natural selection and apply it solve real world problems.They utilise selective pressures found in nature to evolve and develop the fittest possible solutions, these selective pressures in the wild may take the form of limited food or diseases which affect certain but not all organisms in a species. These pressures in nature ensure that the fittest and

most capable organism survive and reproduce in GA's this pressure is emulated using a fitness function to remove the weak and preserve the strong in population.

All GA's share common characteristics and functions they all contain some concept of fitness and thus also have a fitness function to evaluate that fitness. They all use some partner selection method to determine the best parent solutions to recombine to create offspring. They also share the ability to combine two solutions to form a new solution.GA's represent the solutions to a problem as the genes of an organism which can be mutated, modified by recombination to. The representation of these genes can vary, in simple cases and for the purpose of this project they are a string of ones and zeroes but more complex problem can use more convoluted methods for representing genes.

Generation 0	**Generation 5**	**Generation 10**
010101010	010101110	010111111
1010110010	1010110010	1010111110
0110101011	0111101011	1110111111
0100101101	0111101101	0110101101
1011110110	1011110110	1111110111
0101101100	0101101100	0111111110
0110011101	0111111010	0111111101
0101010010	0101011111	1101011110
0001010100	0001010111	1111111111
1100001010	1111001010	1111111010

Figure 2.1 Shows sample generations and their solutions

An important part of the setup and proper function of a GA is the initialisation of the starting population. These populations contain randomly generated solutions which will be used as the starting point of the GA. GA's can utilise a number of population characteristics such as infinite population sizes which do not cap the max population of a generation, they also may use finite population sizes where they do cap the maximum population size. The generational structure can also be modified to suit the problem, overlapping generational models which allow parent solutions to compete with their children for resources and partner selection can be implemented, non-overlapping generational models have often used also where each generation after spawning die off to make way for the next generation. The fitness of each of the population's organisms will be measured by the fitness function of the GA to measure the ability of each. Once this population has been established it can be used to evolve the solutions into fitter solutions the first step of this process is method known as tournament selection , this process is used to filter out the weaker solutions from the population it achieves this by at random selecting a group of solutions and from the population comparing their fitness and selecting the fittest.

Solutions selected by this process will be put into an intermediate generation which will be used to produce the successive generation.

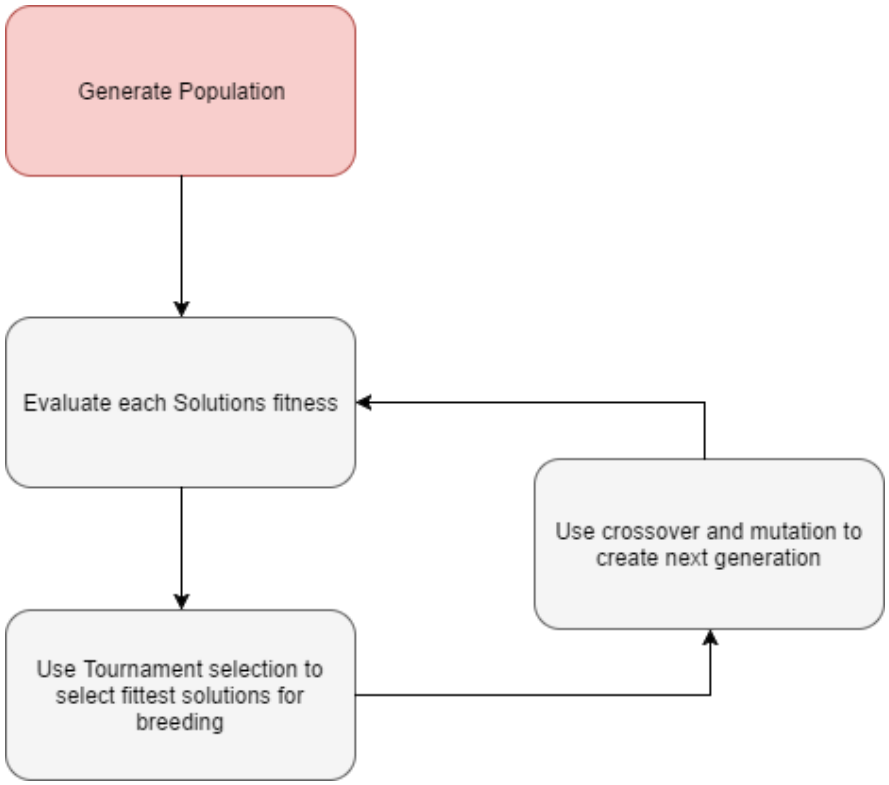

Figure 2.2 Flowchart of Generic genetic algorithm

Each of the solutions of the intermediate generation will gain the opportunity to produce offspring as long as they are fit enough to out compete for the other solutions. Partner selection is the process used to determine the best solutions to recombine. During each recombination there is a small chance of bit flips occurring in the process of copying the genes this is used to imitate the mutation found in natural copying of genes from parents to their children, In GA's this rate can vary and but a common rate is 0.01% meaning that each time a gene is copied it has a 0.01% chance of being copied incorrectly and thus a bit flip occurring. Mutation is a vital part of the evolution of the population it ensures some element of randomness thus allowing unique solutions to take shape which may be fitter and more suitable than other solutions in the population. Modifying the mutation rate can have a large effect on the performance of the algorithm, the mutation rate must be enough to cause a reasonable change in the solution population but not so much that it completely removes any good or useful traits in the population.

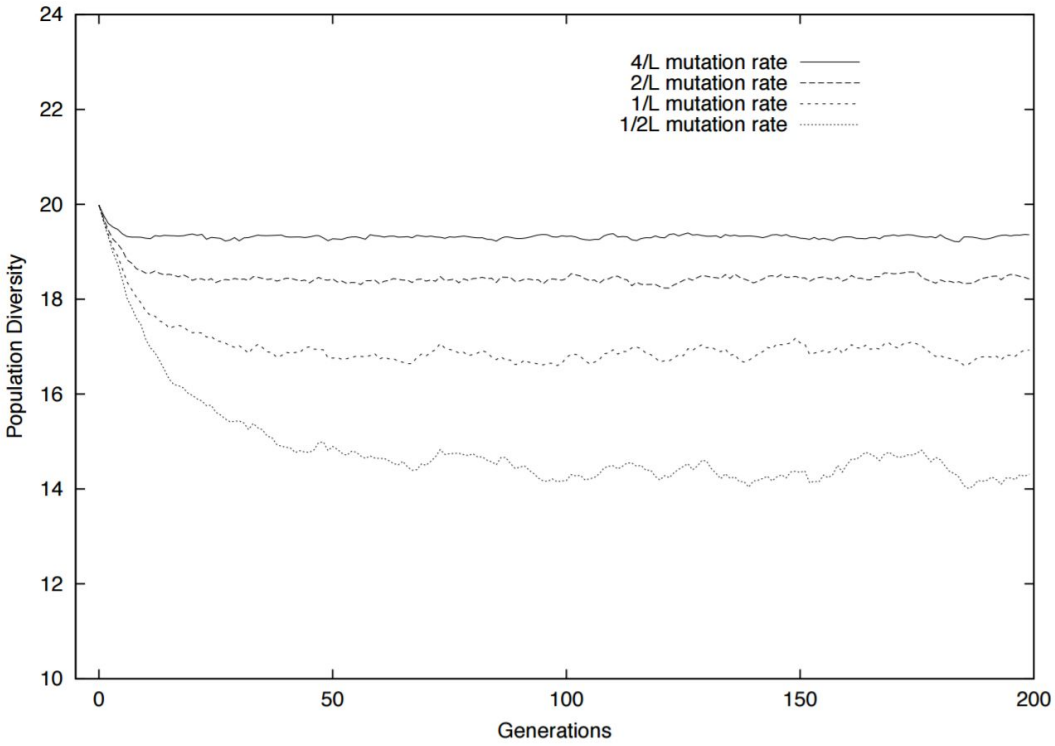

Figure 2.3 Showing effect of mutation rate on Population Diversity L refers to number of Genes in a solution[3]

The diversity of the population is also in large part maintained by the mutation rate, diversity is the degree of difference in a population's genomes.A degree of diversity is vital to health and wellbeing of a population just as in natural systems if there is not a large enough gene pool of potential partners inbreeding can occur and undesirable traits and defects will develop lowering the overall population fitness. After the creation of the next generation is complete the process will start again repeating until a max number of generations has been meeting or the solutions not get any fitter.

2.3 Fitness Function

A fitness function takes in a solution as an input and produces an output[4]. This output is used to define the capability of the solution in solving the select problem.All of the solutions in the population will be assessed using a fitness function allowing the algorithm to determine the best and worst solutions.Often times in large problem spaces problem will be broken out into k subproblems, these divisions in the solutions genes are referred to as *alleles*. An allele is simple to put a number of individual genes grouped together which lead to the expression of a particular phenotype.This feature is added to mimic the alleles found in genetics.Each allele has the potential to be evaluated independently allowing a weighted

fitness function to be used this may be necessary for problem where certain characteristics contribute far more to the overall fitness of a solution than others

$$fitness(g,p) = \sum_{i=1}^{\frac{N}{k}} fitness(g,p_i)$$

Figure 2.4 Fitness function

In figure 2.4 the fitness of a gencode with respect to a problem *p* is the sum of the scores obtained by each *k* alleles in the gencode with respect to a subproblem *pi*.The definition of fitness varies depending on the problem being attempted in simple problems where all genes are represented as ones and zeros a simple sum of all of the genes will suffice in measuring the solutions abilities however in problems containing deceptive elements it is necessary to modify the fitness function to reflect this. One such way of achieving this is by looking for the deceptive landscapes and rewarding an organism fitness for finding it thus modifying the fitness function

2.4 Problem Landscapes

Not all problems are the same, some hill climbing exercises have simple landscapes with one peak others may be more complex having what appears to be a peak but at closer inspection is, in fact, a local optima and is not the global optima.These problems can be considered to be deceptive. Deception in landscapes increases greatly the difficulty the GA will have in finding the optimum solution

Table 1: Fitnesses for non-deceptive function

111	f_1
101	f_2
110	f_2
011	f_2
100	f_3
010	f_3
001	f_3
000	f_4

Table 2: Fitnesses for deceptive function

000	f_1
111	f_2
101	f_3
110	f_3
011	f_3
100	f_4
010	f_4
001	f_4

Figure 2.5 Problem landscapes

In order to effectively test the partner selection algorithm, it is important to test it on a multitude of problem landscapes. Landscapes containing multiple alleles with a mixture of deceptive and non-deceptive are referred to as being partially deceptive problems, where all alleles in the landscape are deceptive these problems are known as fully deceptive landscapes. The varying of allele sizes also leads to a more adverse landscape making it harder for an algorithm to optimise the problem at hand.

2.5 Crossover

Recombination is the process of combining two or more parent solutions to produce offspring and thus create offspring. One way to achieve this is to use a crossover algorithm. Crossover is found in many GA's as it is an easy and effective way of imitating the copying of genes which occurs in reproduction. There are several forms of crossover some of which are.

Single point crossover this is the most basic form of crossover it involves selecting a point at random on the genomes of the parents and selecting all genes to the left of this point from parent A and selecting all genes to the right of this point from parent B combining the two substrings of the genome to give us the genome of the child. The use of one crosspoint preserves large sections of the parents which can be advantageous in that preserve good traits of the parents but the flipside is also true in that it could lead to the inheritance of unwanted or unfit traits.

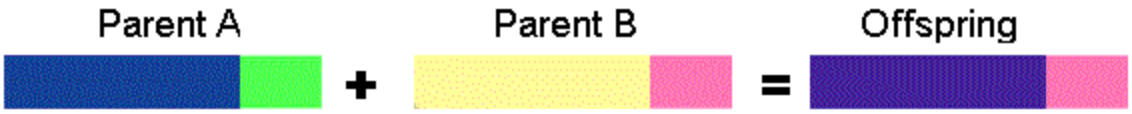

Figure 2.6 Single point crossover[5]

Multi-point crossover is similar to single point crossover but instead of using one cross point it uses multiple crosspoints. The process of creating is similar to above where two at random some points are selected and the genes of the parents are split at these points leading to multiple substrings of genes of the parents. Recombination is then applied to these substrings from the parents to create the offspring.

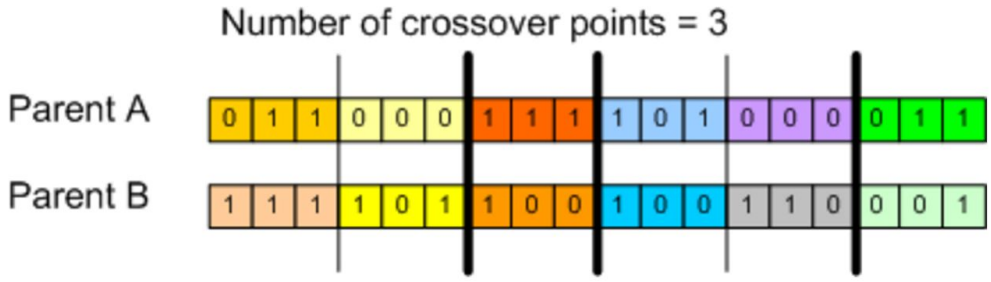

Figure 2.7 MultiPoint Crossover[6]

Uniform crossover, unlike the previous two methods, uniform crossover does not split the parents at points instead it randomly decides between parent A and parent B and selects an allele from them to be used in the child, it repeats this process until the genes of the child are full. These leads to the child comprising approximately 50% genes from parent A and 50% genes from parent B[7].

Parent A: ABCDEF
Parent B : abcdef
Child: AbcDeF

Figure 2.8 Uniform crossover

2.6 Diversity

Small populations of species can suffer from a lack of diversity in their gene pool this leads to problems of inbreeding and the amplification of defects in genes. In GA's a lack of diversity similarly leads to weakness in population, in the case of GA's it leads to suboptimal convergence this is when the solutions of a new generation do not outperform the previous generation[8]. The diversity of a population is often a critical measure of the performance of a GA due to this there are a number of ways to measure diversity of a population. Methods for analysing diversity in *genotypic* space and *phenotypic* space are used. *Genotypic* space refers to the solutions gene's, *phenotypic* refers to the properties or traits expressed by those genes in relation to the fitness function. One measure in phenotypic space is to use a pair-wise and “column-based” measure which measures the variation in values for each phenotypic value [9]

$$H = \sum_{j=1}^{j=P-1} \sum_{j'=j+1}^{j'=P} \left(\sum_{i=1}^{i=L} |y_{ij} - y_{ij'}| \right)$$

Figure 2.9 Example of column based measure

More frequently used than phenotypic measures are genotypic measures.These metrics are more accurately able to quantify the diversity of the population as they measure the solutions at a gene level, one such method for this measure is a pairwise hamming distance.

$$\binom{P}{2} = \frac{P^2 - P}{2}.$$

Figure 2.11 Pairwise hamming measure

It is thus important to attempt to maintain diversity in population. There exists several methods to maintain the diversity of the population. One such solution is to use *elitism, elitism* maintains diversity by selecting the best solutions for the generation and allowing them to go straight into the next population without any competition or any crossover or mutation. Allowing these solutions through ensures that some of the genetic diversity of the previous generation is maintained in successive generations. Injection in GA's involves the injection of new solutions into the population, this injection will occur after there has been no increase in the overall fitness of the population or if there has been little to no variance in the sample population's diversity. Injecting these new solutions increases the diversity of the population and may increase the overall fitness of the population.[8]

2.7 Partner Selection

The method for the selection of partners in nature can vary greatly by species and habitat. These variations are a response to the environment in which the organism has developed, they have been developed through trial and error over millennia to maximise the number of offspring they produce in a lifetime. In GA's these methods of partner selection are mimicked to try to improve the performance and success of the algorithm. Both two partner selection methods and multi-partner selection methods can be used in GA's to produce offspring each of which has their merits, multi-partner selection having the advantage of a larger gene pool to choose from while using parents is simpler and easier to implement. The wide variety of selection methods in nature means that there is also quite a large number of selection methods in GA's which imitate these. The two main families of selection models are Non-overlapping generation models and overlapping generation models.

2.7.1 Non- overlapping generation models

Non-overlapping selection models are selection schemes where parent and offspring do not compete for survival. Once a new generation has been created the previous generation will be removed freeing up resources for the new generation similar to how salmon die after spawning. This strategy has lead to development of several methods for selection process.

Uniform Selection chooses solutions based solely on their occurrence in the population. For example, if all the organisms(solutions) had the same fitness and there were no duplicate solutions then each would have an equal opportunity to produce offspring in the next generation. This process in essence randomly selects two parents to produce offspring. This approach has the advantage of being quite simple to implement as it is just random pairing it does have its downsides one of which is a phenomenon called genetic drift. Genetic drift is the loss of particular genomes from the population due to the chance of some solutions not producing any offspring. Uniform sampling can often lead to the loss of diversity of a population and will eventually lead to the convergence of population to homogeneous fixed point. This defect can be mitigated by increasing the population size but this would also lead to the algorithm requiring more computational effort

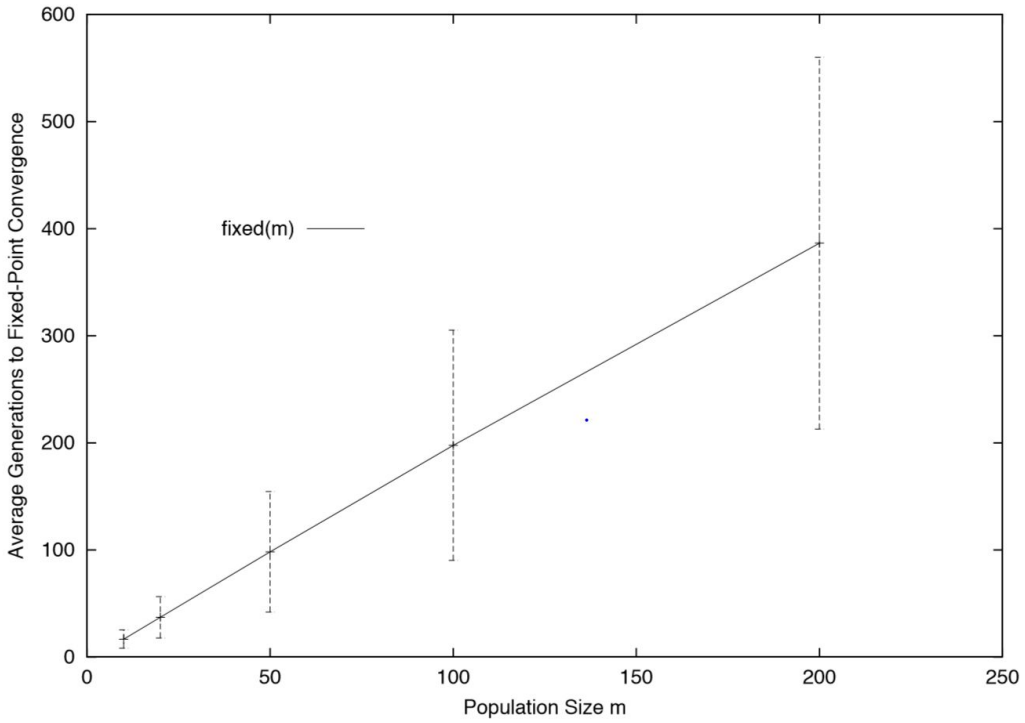

Figure 2.12 Convergence time due to drift with fixed population size m

Fitness Proportionate Selection whereas uniform selection based its selection process only on the rate of occurrence a genome fitness based selection uses a pseudo-random selection process whereby the greater the fitness of the solution the higher the likelihood it will be selected. The usefulness of this method is evident selecting the best solutions to pair and produce offspring very often lead to an increase in the fitness of their offspring and thus a fitter overall population. This approach also has the benefit of adapting to the fitness of the solutions as the population evolve allowing it to more accurately choose the best solution.

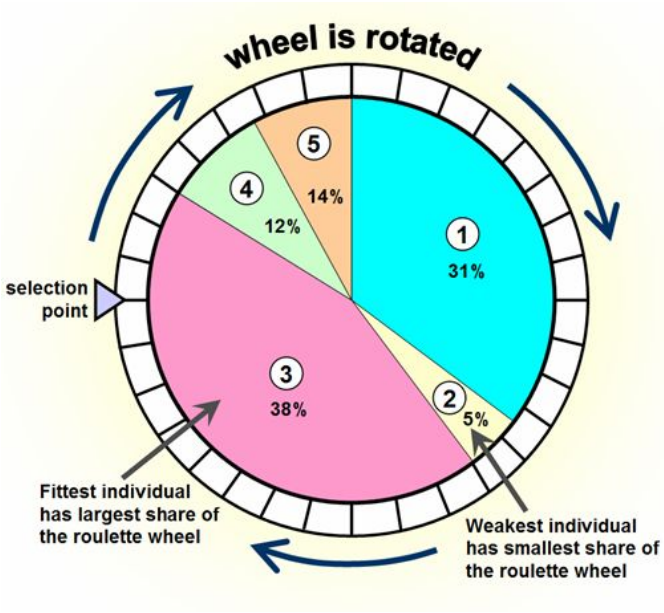

Figure 2.13 Roulette wheel depicting fitness proportionate selection [10]

Tournament selection uses a randomly selected pool of potential solutions to compete against each other selecting the fittest from the pool to mate with another solution. This process again uses the fitness of the population to determine the most appropriate solution to use however unlike fitness proportionate selection it has no random element instead selection is entirely based on the fitness of the solution, this has the benefit of always selecting the fittest solution from the tournament. The choice of tournament size can affect the performance of this method in general, the use of a larger tournament size increasing the overall fitness and diversity of the population.

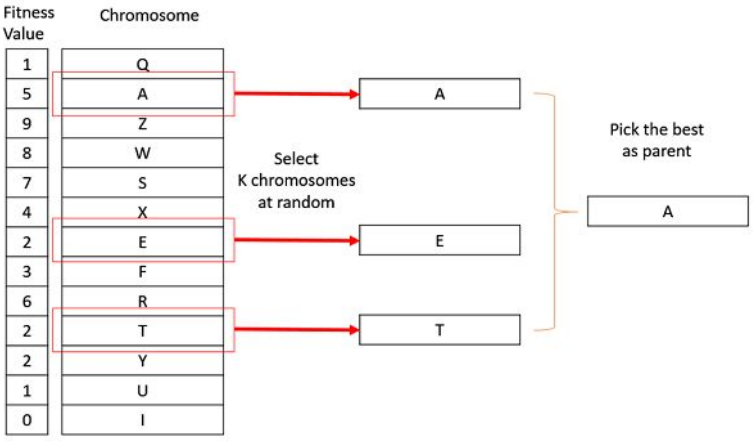

Figure 2.14 Diagram of tournament selection [11]

2.7.2 Overlapping-Generation Models

In many problems it necessary to use an overlapping of generations in order to improve the performance and the convergence rate of the GA on a select problem. The use of this method would then mean non-overlapping partner selection methods would need to be changed in order to accommodate this, because of this requirement we also have partner selection algorithms for these problems with overlapping.

Uniform Selection in overlapping generational models is similar to how it operates non-overlapping models the principal is the same in that it selects two parents from the population at random and use crossover to produce new offspring. However when using overlapping generations with a finite population size it necessary to alter the survival characteristics of the algorithm in order to maintain a consistent overall population size.

2.8 Epistasis

The vast array of variations in species across the biological world is immense, even more so when considered that all of them contain the same building block DNA. One example of this is the similarity in humans DNA to that of chimps, 96% of the DNA is common to both chimps and humans yet the biological differences are vast. One reason for this is the interlinking of nonallelic genes in the expression of traits and features this attribute is called epistasis. In relation to GA's, this increases the difficulty of the problem at hand, attempting to optimise a problem which has interlinking subproblems leads to the need to trade off some problems for others. The difficulty caused by this is evident when considered with a problem which has a deceptive landscape in order to optimize such a problem would be near impossible as the overlapping genes would need to fill both criteria for a deceptive and non-deceptive problem something which is not possible it is thus necessary to come to a compromise for a best all-round solution to the problem.

Design and Implementation

This chapter will contain a detailed account of how the GA was built for this project. It will provide clarity on the innerworking of the program as well as relating them to their concepts in the theory of GA's. The details and logic behind why certain design choices were made will also be discussed. This section will show how, at every stage the system operates from the creation of the population to the partner selection methods.

3.1 Population Creation

The first phase of an Evolutionary computation algorithm is to generate the initial test population. This test population will be a total of one hundred solutions. This populate() function uses a for loop to run from 0 to 99 at each iteration adding solution to the list of organisms called GENERATION, which is the list used to store the current generation of the algorithm. After each organism has been added another loop will run to fill the organism's genes with a random array of zeros and ones' , this is achieved by randomly getting a number between one and ten, if this number is above five a one is added to genes array else, a zero is added.

```
public static List<Organism> GENERATION = new List<Organism>();

public static void populate()
    {
      GENERATION.Clear(); //Clear any previous organism in population
      for (int x = 0; x < POPULATION; x++)
      {
        GENERATION.Add(new Organism(GENES)); // Add empty organisms

        for (int y = 0; y < GENES; y++)   //Randomly generate genes for new organism
        {
          var ran = GetRandomNumber(1, 10);
          var num = ran > 5 ? 1 : 0;
          GENERATION[x].genes.SetValue(num, y);
        }
      }
    }
```

Each of the organisms added to the list is an object created to hold the solutions genes fitness and other properties. This was done to make the data about each solution more accessible as well as allowing the addition of extra properties easier to add at a latter stage.

```
public class Organism
{
    public int fitness { get; set; }
    public List<int> phenotype { get; set; }
    public int[] genes { get; set; }
    public int numberOfdeceptives { get; set; }

    public Organism(int Genes)
    {
        genes = new int[Genes];
        phenotype = new List<int>();
    }
}
```

3.2 Problem Landscapes

The wide range of applications to which GA's can be applied required the development a number of landscapes to test the GA on. In order for the code fulfill this requirement it had to be able to adjust depending on the input parameters. On a non-deceptive landscape no real change in the code was required as the system would just need to optimize the a solution to to contain all one's, however several modifications were required to adapt the system to work work with deceptive landscapes.

The first thing to do when adding the deceptive landscapes was to determine in which allele would they be, to achieve this I added a for loop at the start of the program which would loop and fill the deceptive locations array with unique locations, each of these locations would then be used as a deceptive allele this was repeated until the required required amount of deceptive locations wa added.

```
var numOfDec = GetRandomNumber(1, ALLELES);
numberOfDecptiveLandscape = numberOfDecptiveLandscape > 0 ? numberOfDecptiveLandscape: numOfDec;
            while (deceptiveLocations.Count != numberOfDecptiveLandscape)
            {
                var loc = GetRandomNumber(0, ALLELES);
                if (!deceptiveLocations.Contains(loc))
                {
                    deceptiveLocations.Add(loc);
                }
            }
```

Once these location had been added the program could use these locations to detect the presence of deceptives in the solutions genes.

3.3 Fitness Functions

As mentioned in Chapter 2 the fitness function used to evaluate the fitness of the function of the organism against the landscape on which it is being tested. All the functions used to asses the solutions are explained in this section. These evaluation functions are run at the inception of the initial population and again after every new generation, this is done to avoid this process having to be repeated again at a

latter stage in the program, also because the values assessed by these functions will be used by many of the methods later on in the program.

3.3.1 Fitness Sum

Due to that fact that this fitness function would have evaluate all variations of the landscape I had to some parameters to ensure the correct method of calculating fitness was used, the main discrepancy in fitness measures was when epistasis was being used.

When epistasis is enabled the evaluation of the fitness is modified to count the value of the extra shared bits between the alleles, in this case each of the organisms has to split up into its constituent alleles and the sum of the all the ones in each the alleles would be used as the fitness, the number of alleles would change from ten to eight if the varying allele sizes was in use, the number of genes in each allele would also change from all alleles having three to combination of fives and threes. In the case where epistasis is not used the function would simply sum all of genes of the solution and this would be used as the fitness of the solution.

```
public static void getFitnessAll()
    {
        foreach (Organism org in GENERATION) //Loop for all organisms in the generation
        {
            org.fitness = 0;
            var score = 0;
            if (!useEpistasis)
            {
                score = org.genes.Sum();    // Sum all genes
            }
            else
            {
                var currStartCut = 0;
                var endCut = 0;
                for (var index = 0; index < ALLELES; index++) //Split into alleles
                {
                    List<int> gene = new List<int>();
                                    // If odd overlap the one bit with next allele
                    if (index > 0)
                    {
                        currStartCut = (index % 2 != 0) ? endCut - 1 : endCut;
                        endCut = currStartCut + alleleSizes[index];

                    }
                    else
                    {
                        endCut = alleleSizes[index];
                    }

                    gene = org.spliceGenes(currStartCut, endCut);
                    score += gene.Sum();
```

```
            }
        }
        org.fitness = score; //Assign fitness to solution
    }
}
```

3.3.2 Deceptive Detection

In order to evaluate whether a solution had any deceptive landscapes another function was used, *checkDecptives* was called after the fitness evaluation, if deceptive landscapes were in use. This function similar to the standard fitness function would evaluate all organism in the population and determine how many deceptive landscapes each of the solutions had, it would then attach this number to the organism object. It achieved this by splitting the genes of an organism into their alleles and checking each of these alleles to determine if they were deceptive or not then incrementing the deceptive count of the organism object. This code snippet shows how the program determined if it was the allele (called val in code) was a deceptive that is all the bits were zero. It will also check to see if the current alleles landscape is a deceptive one, it achieves this checking to see if x, which is the number of the allele being tested is contained in the deceptive locations array.

```
gene = org.spliceGenes(currStartCut, endCut);
var val = ListToString(gene);
if (deceptiveLocations.Contains(index) && val.Contains("000") &&    alleleSizes[index] == 3)
  {
    org.numberOfdecptives++;
    numWithDecptive++;

  }
else if (deceptiveLocations.Contains(index) && alleleSizes[index] == 5 && val.Contains("00000"))
  {
     org.numberOfdecptives++;
     numWithDecptive++;
  }
```

3.3.3 Phenotypic Profile

A third evaluation function is called before the tournament selection process can occur this function is subject to phenotypic matching being active, this function is called *getPheno*. The *getPheno* function evaluates all of the solutions in the population assigning it a phenotypic profile based on the value in each of the alleles. To create the profile the function takes the values in each of the alleles which are a string of ones and zeros, and converts this string into a decimal value. Since the program allows the varying of the allele sizes these string can be either of length three or five. Another job of this function is to provide a phenotypic reward if one of the alleles in the solution contains a deceptive, this is a special case captured in if statement. When the condition is true the value pheno value assigned to that allele will be the highest possible value for that length allele.Below it the code showing the allele being

converted into a binary value and this value being added to the phenotypic profile of the solution. The profile created will be used in the partner selection process later on in the program.

```
gene = org.spliceGenes(currStartCut, endCut);
var key = ListToString(gene);
var val = Convert.ToInt32(key, 2);

if ((deceptiveLandscape) && (deceptiveLocations.Contains(index)) )
  {
    if (key == "000")
       val = 9;
    if (key == "00000")
       val = 33;
  }
```

3.4 Tournament selection

After the creation and measurement of the organism was completed the tournament selection process could take place. Tournament selection is the process used to apply a “selective pressure” to the population removing the weak solutions and maintaining the fit ones. To implement this in code is relatively simple the first thing is to randomly select some number of solutions, this group randomly selected organisms will be a tournament. After the tournament is populated to a predetermined tournament size you select the fittest organism to be added to the breeding population, the fitness is simply the fitness of the solution added to the number of deceptives of the solutions contains times the deceptive reward, which is constant a used to increase the fitness of the solution for finding deceptives.

```
for (var a = 0; a < compSize; a++)
{
              var sel = GetRandomNumber(0, POPULATION);
              score = 0;
 score = (GENERATION[sel].numberOfdecptives * deceptiveReward) + GENERATION[sel].fitness;
              if (score > bestScore)
              {
                 bestScore = score;
                 bestPerformer = sel;
              }

 }
generationBestPerformer = generationBestPerformer < bestScore ? bestScore : generationBestPerformer;
tempGen.Add(GENERATION[bestPerformer]);
```

In the sample code above the the tournament size, called comp size in the code is set to four, so four organisms will be in a tournament. The fittest from these four will be added to the temporary population used in the creation of the next generation, this newly created generation will be passed as a

parameter into the next function called *generateNextGen* this method handles the creation of the next generation.

3.5 Partner Selection

Before the production of offspring can occur two parents in the temporary population have to be paired, this is done using the *getPartner* method, this method has a number of ways to match two parents all of which use a tournament selection like process and a difference measure to find the best match. The selection process is run over all of the solutions in the population meaning all solutions will produce at least one child with particularly fit solutions having a high probability of producing more than one.

3.6.1 Fitness only

In the standard partner selection process two measurement metrics are used, these metrics being the solutions fitness and the number of deceptive the solutions have. Similar to tournament selection a number (in our case four) potential partners will be selected at random from the population each being tested to see if it is the most suitable. In fitness only this means it is effectively another tournament selection.

3.6.2 Hamming

When the partner selection process is using the hamming measure a slight modification the partner selection algorithm is made, this being the addition of a hamming difference calculation between the two potential partners. As in the standard partner selection process a group of four potential partner is randomly chosen from the temp population, each of these are measured against the solution the program is currently finding a partner for. The value returned from the *getHamming* distance function is then added to the fitness of the solution to get is score, this score is what is used to determine the most eligible candidate for the solution. The code below shows how the score is calculated . Fitness weight and Phenotypic weight are used to weight the values of the fitness and phenotypic difference respectively, these are set to one in the GA as this was the best performing. Phenotypic weight is multiplied by the diff value which in this case, is the hamming distance .Deceptive reward is the value used to increase the score of the solution for containing the peak value in a deceptive allele. This process is repeated for all of the solutions in the tournament selection pool.

```
if(useHamming)
      diff += getHammingDistance(orgA, orgB);

score = ((fitnessWeight * orgB.fitness) + (orgB.numberOfdecptives * deceptiveReward )) + (phenotypicWeight * diff);
```

3.6.3 Phenotypic matching

Another method built to find best match for a solution is that of the phenotypic matching process. This process would use the phenotypic profile made earlier in the program by the *getPheno()* method. This in

essence is the same process as the hamming partner selection with the exception being that the diff value would be obtained from the *getPhenoDifference()* method. This method would measure the distance of the phenotypic profiles by subtracting each of the phenotypes from each other and normalizing them(shown in code below). The allele sizes array is just an array containing each of the allele sizes, a being the current phene being checked. Once this difference value is returned it used to assess the eligibility of the current pairing, the process is repeated for all of the solution in the test pool.

```
float diff = 0;
diff = Math.Abs(orgA.phenotype[a] - orgB.phenotype[a]);
float div = alleleSizes[a] == 3 ? 3 : 5;
score += diff/div;
```

3.6 Crossover

The process of combining the parents in the temp population created in tournament selection process occurs next in the program. Crossover is the process of combining the genes of two or more parents to produce a new solution. There are a number of methods used to accomplish this, some of these include single point crossover, multipoint crossover and uniform crossover. A number of these methods have been added to GA to help better test and understand the affect each partner selection has on the performance of the GA.

3.5.1 Single and Multi Point Crossover

The simplest form is single point crossover, as shown in chapter two it uses one point on which to cross the genes, all genes less than this point are added from one parent while all genes greater than this point are taken from the other parent. To implement this feature in code is straightforward enough but, when using the same crossover function to do all types of crossover things get a little more complicated. The code in the following snippet shows the working of the method. Firstly a random point is chosen to be the crosspoint, this value is then added to the crosspoints list. The loop will iterate for however many crosspoints are in use, for each of these crosspoints a gene splice will occur this will remove the genes from the parent in the select range, following this these spliced genes will be added to the new solution (stored in next gen). Crosspoints will always contain at least two values, the first being a zero value used to start the cut e.g if the random cross point is thirteen then the first cut will be 0 to thirteen and the second will be thirteen to thirty (the number of genes in a solution). The parents the genes are spliced from are alternated between cuts. Multi Point crossover uses the exact same piece of code just with more values in the cross points algorithm, this number can be set by the user to any desired amount.

```
for (var b = 0; b < crossPoitns.Count; b++)
{
    endPoint = b + 1 < crossPoitns.Count ? crossPoitns[b + 1] : GENES;
    if (parent == 0)
      {
       gene = gen[pop].spliceGenes(crossPoitns[b], endPoint);
       nextGen[pop].addGenes(gene, crossPoitns[b], endPoint);
```

```
        parent = 1;
        }
      else if (parent == 1)
        {
          gene = gen[parentB].spliceGenes(crossPoitns[b], endPoint);
          nextGen[pop].addGenes(gene, crossPoitns[b], endPoint);
          parent = 0;
        }
}
```

3.5.3 Weighted Crossover

The weighted crossover algorithm again uses the same process by which to add new genes as the single point and multipoint crossover use with the exception that is of how the cross points are chosen. The cross points used in weighted cross over are deterministic whereas the points in single and multipoint are stochastic. To get this points for weighted crossover we need to use the fitness of the parents being recombined we need these to weight the crosspoints in relation the each of the parents fitness e.g if both had the same fitness then the point would split both parent in half. The code below shows this process in action the fitness is found using both the fitness of the solution and the number of deceptives in the solution.

```
float fitnessA = gen[pop].fitness + (gen[pop].numberOfdecptives * 10);
float fitnessB = gen[parentB].fitness + (gen[parentB].numberOfdecptives * 10);
float com = fitnessA + fitnessB;    //Combined fitnesses
float cross = (fitnessA / com);
cross = cross * GENES;                          //Cross point
crossPoitns.Add((int)cross);
```

3.5.4 Uniform Crossover

Uniform crossover allows both parents roughly fifty percent of their genes to any potential offspring, meaning that for each gene added to the offspring has a fifty percent chance of being from parent A likewise from parent B. To implement this all that's needed is a simple loop to iterate over the number of genes in a solution generation a random number each time and using this number to determine to select from parent A or parent B

```
 for (var g = 0; g < ALLELES; g++)
      {
        var rand = GetRandomNumber(1, 10);
         if(rand < 5)
         {
           end += alleleSizes[g];
           var genes = gen[parent].spliceGenes(start, end);
           nextGen[pop].addGenes(genes, start,end);
         }
         else
         {
           end += alleleSizes[g];
           var genes = gen[parentB].spliceGenes(start, end);
```

```
            nextGen[pop].addGenes(genes, start, end);
        }
        start = end;
```

3.7 Mutation

After the crossover had been completed and the new solution it then needed to be mutated, this was the next process to run. The mutation rate used was 0.01 %, this was chosen as it showed good performance, it was not to high to mitigate the effect of the crossover and too low that it ha no effect. Designing the *Mutate()* function meant randomly generating a number between one and hundred, then using this number to mutate a function, this process would occur inside a loop iterating over all of genes in the solution. Once all of the genes had been checked the process would finish.

```
var random = GetRandomNumber(1, 100);
            //Mutation of gene
      if (random == 1)
        {
                if (org.genes[g] == 0)
          {
            org.genes[g] = 1;
          }
         else
          {
            org.genes[g] = 0;
```

3.8 Alleles

Allelic variation would test the partner selection algorithms ability to perform on more complex and less consistent problem. Since the length of genes was thirty and we needed to vary the size of these genes only one size variant was coded, that is one other allele size from the standard size of three.It was decided that a allele size of five would be added, this would mean that when varying the sizes of alleles would lead to a mix of alleles of length five and three. The maintaining of the gene length at thirty and the decision to combine lengths of threes and fives meant that their could only be three alleles of length five and eight alleles of length three when varying the allele size. The order of the allele sizes would be randomized at every new run of the GA and added to an allele size array used to show the sizes of the alleles at each point.

3.9 Elitism

The purpose of elitism is to at some degree preserve the diversity of the population ,although when using a large population size this effect is minimal at best. In order to implement this feature at the start of every generation, that is before the tournament selection process the fittest solution will plucked from the generation and immediately added to the next generation thus allowing it to avoid any of the selective pressure applied to the current generation.

3.10 Results Gathering

Several metrics were used to measure the ability of the GA to solve the problems presented to it such as, the average number of generation it took found the ideal solution the average, best and worst fitness in a generation for each of the generations would also be gathered. At the end of every run of the GA four files would be generated a summary file, a generational fitness file,an average of the times taken and the best average and worst of every generation. In the majority of the experiments each test, that is for each number of deceptive landscapes tested, the algorithm would be run one hundred times this would be done a further one hundred times giving a total of ten thousand runs in each test.

4 Testing and Results

This section will show and explain the results gathered from each of the experiments used to test the partner selection algorithms. Each of the experiments has different parameters all of which will be outlined in the start of the experiments. The experiments are designed to focus on the performance of the partner selection algorithms it a range of problem landscapes testing its performance using a number of metrics.

4.1 Experiment 1: Standard Landscapes

This first experiment is designed to test the performance of each of the three main partner selection process on a number of deceptive and non deceptive landscapes. For each run of the experiment the average runs taken to reach the optimum solution, this would give an idea of the overall performance on each of the problem landscapes.Fail rate will also be calculated to determine that number of times the GA was unable to find the optimum solution.

This experiment would test the algorithm using a varying number of deceptive landscapes.One deceptive landscape will mean that one the ten alleles in the genes will be deceptive i.e the optimum gene sequence for that allele will be a string of three zeros.The position of the deceptive landscapes in will be randomised at the start of each run of the algorithm.Since the allele sizes will be fixed in the experiment each of the alleles will contain three genes. Each individual run of the program consisted of running the GA one hundred times and each time an ideal was found the number of generations taken to reach the ideal was measured, if an ideal was not found in under one hundred generations the GA would stop executing and this would be considered a failure to converge. This process was repeated another one hundred times meaning that the GA be run for a total of ten thousand, these ten thousand runs would would equal one set of results e.g the results for using one deceptive landscape are the results of ten thousand runs of the GA.

Parameters Used:

Population	Gene Size	Number of Alleles	Allele Sizes	Recombination Method
100	30	10	3	Single Point crossover

Figure 4.1 below shows a comparison of the performance of the GA utilizing each of the partner selection algorithms on a varying number of deceptive landscapes. The y axis shows the average number of generations taken to find the optimum solution, this is the average of the ten thousand runs in each of the landscapes. The blue is depicting the fitness only method of partner selection that is only the fitness of the two prospective parents is used to determine if it is a good match, the yellow depicts the use of partner selection using the hamming distance between the potential parents and the fitness to

determine the most suitable pairing. Finally the teal color displays the use of the phenotypic difference between potential partners and the parents fitness to find a best match.

This chart gives a good indication of the performance of each of the partner selection algorithms on each of the problem landscapes showing that the fitness only is the selection method is the worst performing of the three while both the hamming and phenotypic matching provide an increase in performance in all of the problem landscapes. The benefit of using these partner selection methods increase dramatically as landscapes go from partially deceptive to fully deceptive. The performance of the hamming and the phenotypic matching are effectively the same both returning similar overall results.

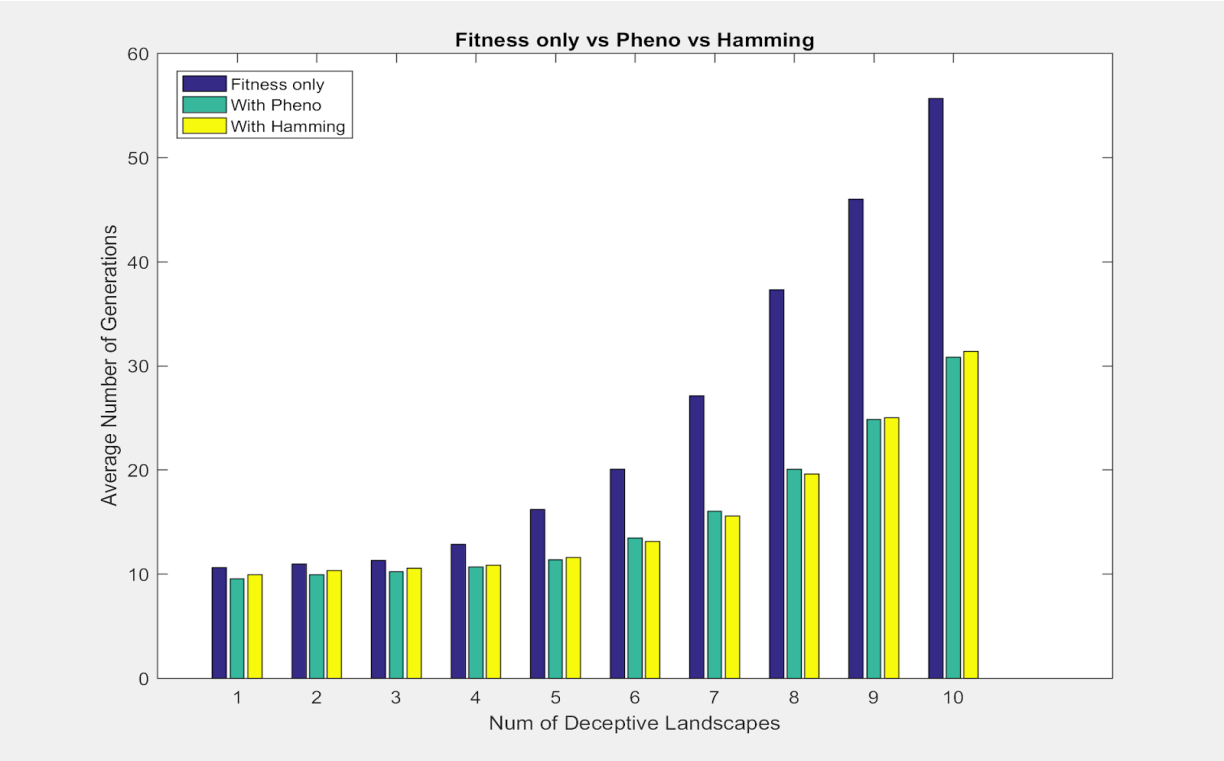

Figure 4.1 Bar Graph comparing different partner selection algorithms

The following two tables show the fail rates and the mean and standard deviation of the three selection methods tested. Similar to the charts above the hamming and the phenotypic outperform the fitness only matching. There is however an interesting pattern in the fail rates of the method when the number of deceptives is under five, this being that the fail rates are higher for hamming and pheno. This may be due to the hamming and pheno somewhat overweighting the value of the deceptive landscapes leading the algorithm astray from the optimum solution, this is occurs in quite a small number of cases and has little to no effect on the overall averages. When using above five deceptive landscapes the hamming and pheno begin to well outperform the fitness only method leading to a dramatic reduction in the percentage of times the GA failed to find the optimum. Figure 4.2 showing the mean and standard deviation further confirms the fact of the pheno and hamming outperforming the fitness only, this table

also shows that when using these two alternative methods leads to reduction in the spread of the results

	Fail Rates %		
	Fitness Only	**Hamming**	**Phenotypic**
1 Deceptive	0 %	0%	0%
2 Deceptive	0 %	0%	0%
3 Deceptive	0.004 %	0%	0.02%
4 Deceptive	0.0222 %	0.26%	0.48%
5 Deceptive	0.0614 %	1.18%	1.37%
6 Deceptive	11.86 %	3.26%	3.94%
7 Deceptive	19.1 %	6.48%	7.22%
8 Deceptive	30.94 %	11.7%	12.26%
9 Deceptive	46.02 %	18.36%	18.38%
10 Deceptive	55.68 %	26.41%	25.87%

Figure 4.1 Experiment 1 Fail Rates

	Fitness Only		**Hamming**		**Phenotypic**	
	Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
1 Deceptive	10.65	1.74	9.94	1.5	9.56	1.5
2 Deceptive	10.99	2.52	10.37	1.63	9.95	1.59
3 Deceptive	11.34	6.11	10.58	1.7	10.23	2.12
4 Deceptive	12.88	13.2	10.85	4.89	10.69	6.5
5 Deceptive	16.23	21.52	11.58	9.84	11.4	10.6
6 Deceptive	20.1	29.06	13.13	16.14	13.47	17.6
7 Deceptive	27.1	35.46	15.58	22.31	16.06	23.55
8 Deceptive	37.3	42.1	19.63	29.38	20.06	29.9
9 Deceptive	46.02	45.2	25.02	35.6	24.87	35.7
10 Deceptive	55.68	46.58	31.4	41.2	30.85	40.96

Figure 4.2 Experiment mean and standard deviation

4.2 Experiment 2: Varying Allele

One way to add more difficulty to the problem landscapes is to vary the allele sizes, which is what we will be testing in this experiment. As explained in chapter three due to the fact that there is a fixed number of genes in testing the degree to which the allele sizes can vary is limited, and so because of this the allele sizes will be either three bits long or five bits long in this experiment, adding to this number of alleles containing five bits is limited to three while their are five alleles containing three bits the order of the allele sizes is randomised on every run of the program.In this experiment the number of alleles is set to eight instead of the usual ten. The number of deceptive landscapes will again be varied from non deceptive to fully deceptive, fully deceptive in this case will mean having eight deceptive landscapes.

Parameters Used:

Population	Gene Size	Number of Alleles	Allele Sizes	Recombination Method
100	30	8	3 and 5	Single Point crossover

This first bar chart shows the average number of generations taken for the GA to find the optimum solution for using each of the partner selection methods. When using the varying allele sizes and the deceptive landscapes the average generations taken find the optimum is significantly higher than when using fixed allele size for example the average generations taken to find the optimum when using three deceptive landscapes and fitness only matching in experiment one (using a fixed allele size) was 11.34 while in this test it takes nearly double that at 20.2 showing the increased adversity of this problem set .

Despite this increase in the complexity of the problem landscape both the hamming and the phenotypic matching continue to provide an increase in performance. The benefits provided by using both the hamming and phenotypic matching are evident, they also provide similar enhancements to the algorithm reducing the average generations taken across all test cases. Figure 4.4 which displays the fail rates of each of the partner selection methods shows a reduction in the number of fails when using the alternative selection methods over the fitness only method the rate of this reduction increases as the difficulty of the landscape increases. Finally figure 4.5 shows the mean and standard deviation of the results and again the results show an improvement when using the hamminga and phenotypic matching in both mean and the standard deviation that is with the exception of the standard deviation when using a large number of deceptive landscapes, this is due to the fact that the mean is high and the max number of generations allowed is one hundred leading to the standard deviation being lower than it is in hamming and deceptive in the same problem, this can be shown by the fact the fail rate at this level is 75%, meaning that 75% of the runs produce an output of one hundred.

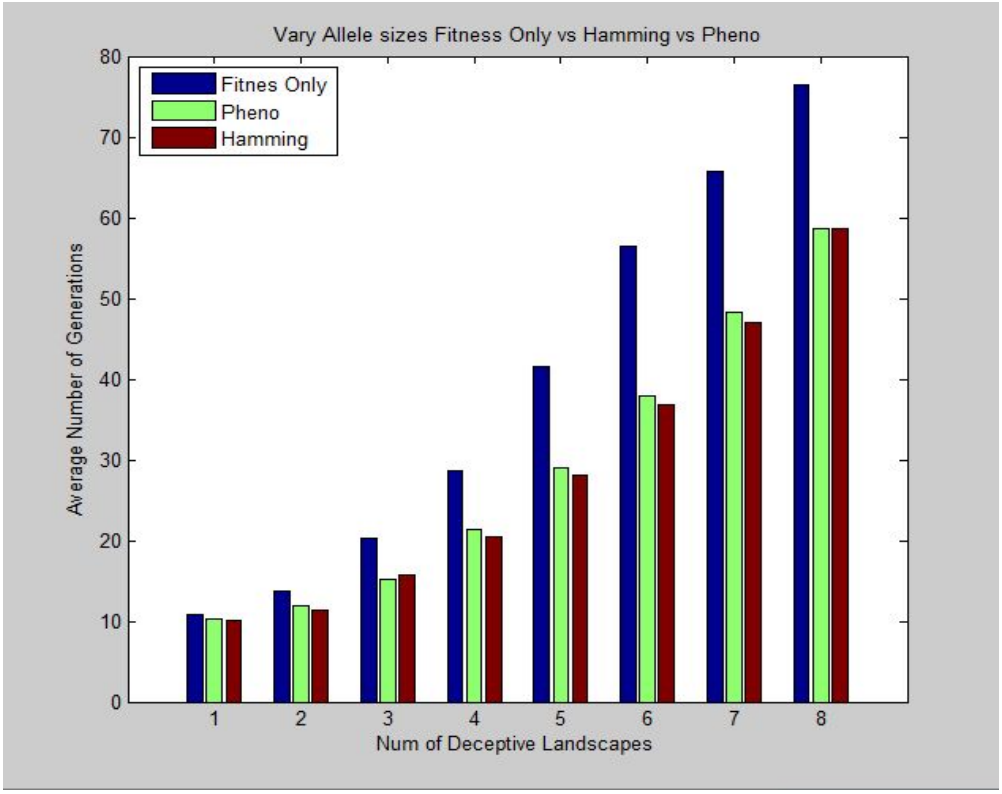

Figure 4.3 Bar chart of Vary allele size for different partner selection methods

	Fail Rates %		
	Fitness Only	**Hamming**	**Phenotypic**
1 Deceptive	0.16 %	0.13%	0.12%
2 Deceptive	3.27 %	1.74%	1.02%
3 Deceptive	10.31 %	5.11%	5.73%
4 Deceptive	21.26 %	12.14%	11.32%
5 Deceptive	34.99 %	20.82%	20.05%
6 Deceptive	51.84%	31.55%	30.21%
7 Deceptive	62%	43.63%	42.2%
8 Deceptive	74.97 %	55.79%	55.74%

Figure 4.4 Fail rates for Vary allele size using different partner selection methods

	Fitness Only		Hamming		Phenotypic	
	Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
1 Deceptive	10.8	3.98	10.2	3.61	10.1	3.51
2 Deceptive	13.8	15.93	12	11.83	11.3	9.16
3 Deceptive	20.2	27.12	15.2	19.75	15.7	20.85
4 Deceptive	28.7	36.55	21.4	29.27	20.5	28.4
5 Deceptive	41.6	42.83	28.94	36.49	28.1	36.01
6 Deceptive	56.4	45.23	38	42.18	36.8	41.57
7 Deceptive	65.8	44.36	48.3	45.36	47.04	45.29
8 Deceptive	76.5	40.54	58.66	46.44	58.59	46.47

Figure 4.5 Fail rates for Vary allele size using different partner selection methods

4.3 Experiment 3: Uniform crossover

The previous experiments may have changed the landscape but all used the same crossover method, that is all used the single point crossover method.In this experiment the uniform crossover method will be tested. As explained in previous chapters uniform crossover selects an allele from parent A or parent B at random and adds it to the child this process is repeated until all of the alleles have been added to the offspring. UC (uniform crossover) is low disturbance operator meaning that it does not split and mix the genes as well as other crossover methods. Same as previous experiments the performance will be tested by calculating how long it takes the GA to reach the optimum solution on a number of deceptive landscapes.

Parameters Used:

Population	Gene Size	Number of Alleles	Allele Sizes	Recombination Method
100	30	10	3	Uniform crossover

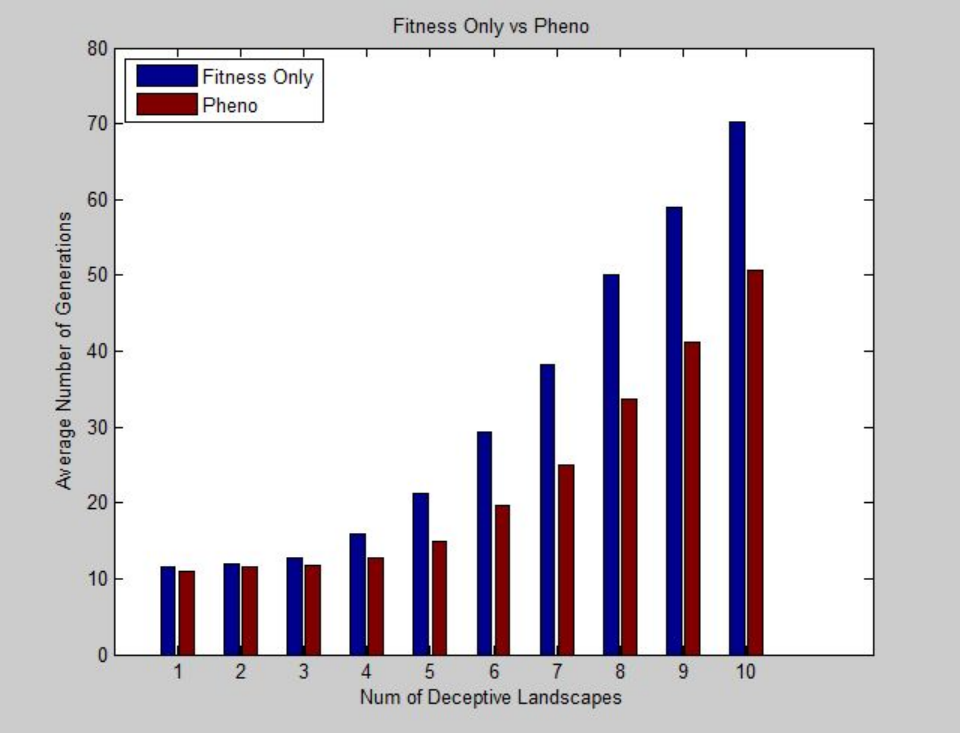

Figure 4.6 Uniform crossover Fitness only v Phenotypic matching

	Fail Rates %	
	Fitness Only	**Phenotypic**
1 Deceptive	0%	0%
2 Deceptive	0.08%	0%
3 Deceptive	1%	0.14%
4 Deceptive	4.81%	1.42%
5 Deceptive	11.21%	4.24%
6 Deceptive	20.7%	9.86%
7 Deceptive	31.29%	16.34%
8 Deceptive	44.8%	26.7%
9 Deceptive	56.4%	36.03%
10 Deceptive	68.2%	47.25%

Figure 4.7 Uniform crossover Fitness only v Phenotypic matching

	Fitness Only		Phenotypic	
	Mean	**Std Dev**	**Mean**	**Std Dev**
1 Deceptive	11.6	2	11	1.91
2 Deceptive	11.94	3.3	11.4	2
3 Deceptive	12.8	9	11.75	3.9
4 Deceptive	15.9	19	12.71	10.6
5 Deceptive	21.26	28	14.98	18
6 Deceptive	29.3	36.2	19.64	26.7
7 Deceptive	33.8	41.7	24.89	33.3
8 Deceptive	50	45.25	33.7	40.2
9 Deceptive	59	45.75	41.28	44.75
10 Deceptive	70.2	43.71	50.6	46.8

Figure 4.8 Uniform crossover mean and Standard deviation

The results from these experiments are in line results from previous experiments where the use of phenotypic matching improves the performance of the algorithm across the board even when using the uniform crossover method. Using phenotypic matching lowers the average number of generations taken to reach the optimum and reduces the fail rates for each test, this benefit seems to get more pronounced the greater the number of deceptive landscapes.

4.4. Experiment 4: Epistasis

In the two previous experiments all of the landscapes had a global optimum which made it relatively easy to measure the ability of the GA on these landscapes, this was done by just finding how many generations it took to reach this ideal solution however, in an epistatic problem landscape there is no ideal solution in the same way there is in the previous experiments. In experiment one and two the ideal was just the optimization of all the alleles in a landscape and as the alleles were unlinked changing one did not affect the performance of another allele in the solution however when using epistasis there is a link between the alleles as they share bits in common so if one allele is changed it can have an effect on other alleles in the solution. This interlinking in this experiment meant that there was no ideal in the same way there was in previous experiment meaning that different ways of quantifying the performance of the algorithm has to used.

Parameters Used:

Population	Gene Size	Number of Alleles	Allele Sizes	Recombination Method
100	25	10	3	Single Point crossover

The charts below show the rate of convergence of the GA, since there is no ideal these graphs measure the rate at which the GA can reach a steady state solution that is where the fitness of the solution stays constant and can climb no further. The graphs depict the rate of convergence in a fully deceptive landscape, a partially deceptive landscape and a non deceptive landscape each of which show the use fitness only partner selection (Avg & Best) and the phenotypic matching (Avg Pheno & Best Pheno). The fitness of the organism , on the y axis, from zero to one with one being the max fitness attained in that test, the x axis is the number of generations passed ,capped at fifty in this experiment. The best and the average fitness are measured at every generation and these results are then graphed to produce the charts below.

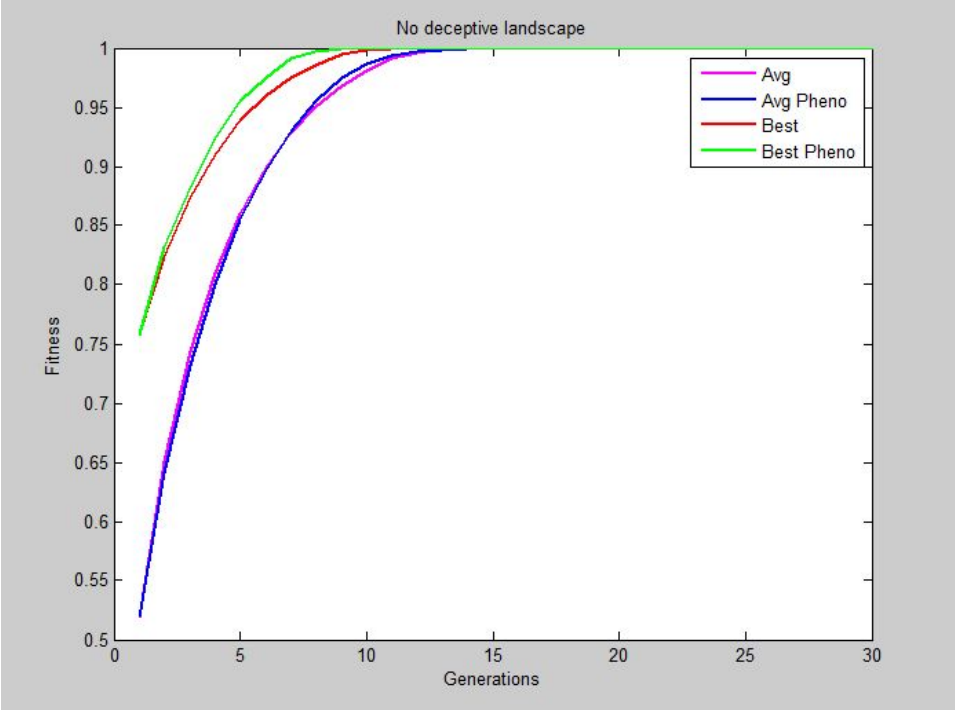

Figure 4.6 Epistasis on non deceptive landscape

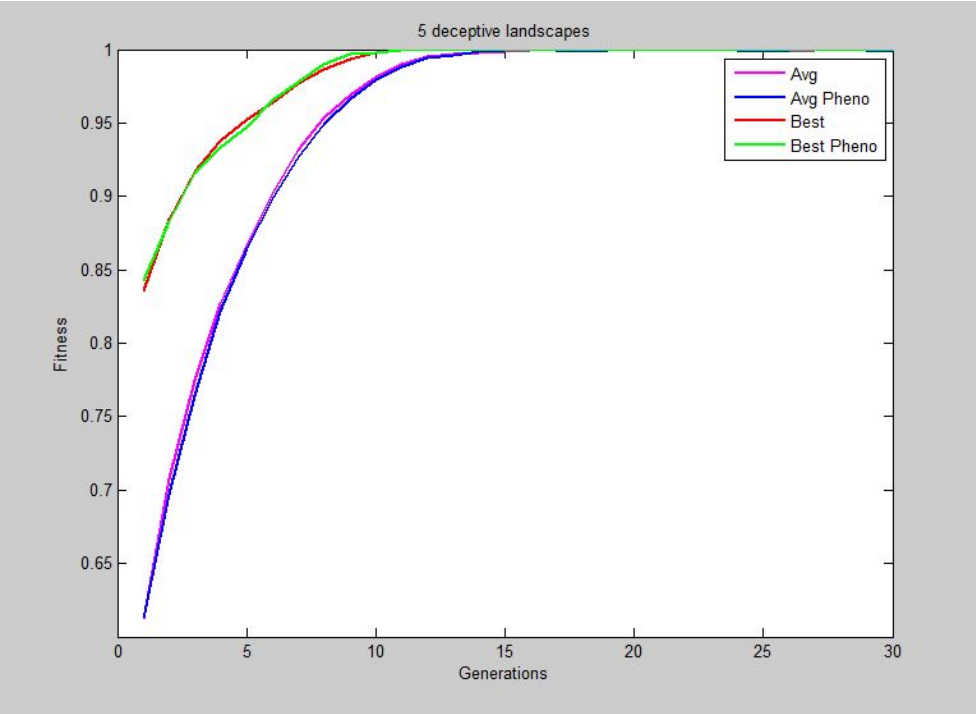

Figure 4.7 Epistasis on partially deceptive landscape

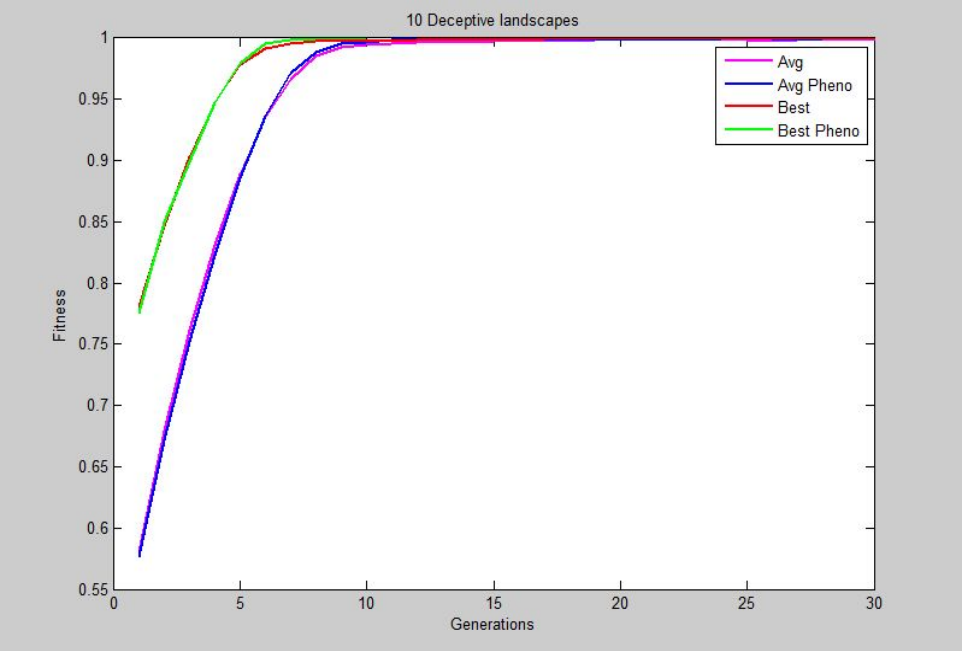

Figure 4.8 Epistasis on fully deceptive landscape

The results from this test show that the use of phenotypic matching does not improve performance in every landscape. As seen in the graphs below the performance boost when using the phenotypic matching in these landscapes is negligible. Across all of the test run in this experiment there was no statistically significant increase in performance of the GA when using the phenotypic matching

5 Conclusion

The aim of this project was asses the performance and traits of using a phenotypic based partner selection process in GA's ,and to determine whether or not it performed better that a standard fitness based partner selection process, as the results show it performs better on most of the landscapes tested in this project. This work demonstrates that with an increase in the adverseness and complexity of landscapes the phenotypic partner selection method continued to deliver an increased performance in the majority of landscapes of, the one exception being when an epistatic landscapes was used . The findings thus prove that the implementation of a phenotypic matching process can increase the ability of a Genetic algorithm to solve and optimize a problem.

5.1 Future Work

There is still quite a number of landscapes that the phenotypic matching could have been tested on as well as several other recombination methods which could have been used, many have which may have provided more detail about phenotypic matching and its strength and weaknesses. For instance, all of the experiments conducted were done on fixed population sizes and non overlapping generations adding these two new properties may have altered the behavior of the GA and the preferences of the partner selection process. Another possible modification would have been to limit the population of the GA and see how the phenotypic matching would fare when using a smaller gene pool. The problem of optimizing a string of ones and zeros is not a particularly complex one other problems such as a travelling salesman problem would prove more difficult to solve for the algorithm

The array and variation of methods of recombination used in genetic algorithms is vast, implementing more of these recombination methods and observing the effect they have on the various partner selection algorithms.The use of the polyamorous relationship would have been another interesting addition, this would mean the use of more than two parents to create offspring giving any potential offspring a greater diversity of genes to choose from and may offer new insight into the properties of the partner selection algorithms. The creation of more than one offspring by the same parents is another potential addition to the project.

References

[1] *A history of evolutionary computation (1997), Kenneth De Jong, David B Fogel and Hans-Paul Schwefel*

[2] Genetic Algorithms Are NOT Function Optimizers (1993), Kenneth A. De Jong

[3] Evolutionary Computation A unified approach,(2006), Kenneth A. De Jong

[4] www.tutorialspoint.com. (2017). Genetic Algorithm's Fitness Function. [online] Available at: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm [Accessed 21 Mar. 2017].

[5] Crossover and mutation. 2017. Crossover and mutation. [ONLINE] Available at: https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/cromu.html. [Accessed 21 March 2017].

[6] Benjamin Hadorn. 2017. Genetic Algorithm. [ONLINE] Available at: http://www.xatlantis.ch/index.php/education/zeus-framework/49-genetic-algorithm. [Accessed 21 March 2017].

[7] FALKENAUER, E. The worth of the uniform [uniform crossover] - IEEE Xplore Document *Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=782011&tag=1 [Accessed 21 March 2017]*

https://www.researchgate.net/figure/267867780_fig1_Fig-3-Population-updating-in-GA-uniform-crossover-operator-and-binary-swap-mutation. [Accessed 21 March 2017].

[8] An Overview of methods maintaining Diversity in Genetic Algorithms [2012] Deepti Gupta , Shabina Ghafir

[9] Measurement of Population Diversity , Ronald W. Morrison , Kenneth A. De Jong

[10] Genetic Programming : Difference between Roulette Rank and Tournament Selection - Stack Overflow. 2017. Genetic Programming : Difference between Roulette Rank and Tournament Selection - Stack Overflow. [ONLINE] Available at: http://stackoverflow.com/questions/23183862/genetic-programming-difference-between-roulette-rank-and-tournament-selection. [Accessed 23 March 2017].

[11] tutorialspoint.com. 2017. Genetic Algorithms Parent Selection. [ONLINE] Available at: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm. [Accessed 23 March 2017].