Department of Information Technology

technical report NUIG-IT-150601

Genetic Algorithms, their Operators and the NK Model

S. Hill (NUI, Galway)
C. O'Riordan (NUI, Galway)

Genetic Algorithms, their Operators and the NK Model

Seamus Hill and Colm O' Riordan,
Department of Information Technology, Galway,
Galway.

Abstract

This paper outlines the operators and workings of Genetic Algorithms, and Kauffman's NK model. To analyse the performance of genetic algorithms and their operators the fitness landscape is crucial. A discussion on fitness landscape is included, which paves the way for Kauffman's NK model to analyse to performance of genetic algorithms and their operators. Future work which may extend from this could include using the NK model to analyse the performance of the inversion operator.

1 Introduction

The book *Adaptation in Natural and Artificial Systems* written by John Holland, presented Genetic Algorithms (GA's) as an abstraction of biological evolution and presents a theoretical framework, which can be adopted for GA's[8]. For Holland, the GA is a method of moving from one population of bit strings which represent creatures or possible solutions to a problem, to a new population, using selection along with crossover, mutation and inversion operators. Given an optimisation problem, the set of possible solutions to this problem can be encoded using strings of a fixed length formed from some finite size alphabet. This encoding generates a representation space, which is a high dimensional space of all possible strings of a particular length, each of which encodes a possible solution to the problem. The effect of the choice of operators to use and their associated rate has been a topic of much debate and research.

To develop a theory regarding the choice of specific genetic operators (i.e. the inversion operator) to include in a GA and the rate at which they should be set, a model such as Kauffman's NK model can be used to analyse their performance. This is because the structure of a fitness landscape depends on the underlying problem, and developing a theory about operators and their probability rates should be independent of the structure of the landscape. To this end, Kauffman's NK model provides a problem independent landscape, as the fitness landscape can be gradually turned from smooth to rugged.

2 An Overview of Genetic Algorithms

Genetic Algorithms (GAs) are search algorithms based on the mechanics of natural selection and natural genetics. They are a combination of survival of the fittest among string structures, and a structured yet randomised information exchange to form a search algorithm which has human-like search qualities. With each generation, a new set of artificial creatures (strings) are created using parts of the fittest of the old, combined with some new parts. Although randomised, genetic algorithms efficiently exploit historical information to speculate simple search points with expected improved performance. GA's can be classified as a group of computational models inspired by natural evolution. A potential solution to a particular problem is encoded on a data structure similar to a chromosome. Each chromosome consists of a number of "genes" (e.g. bits), with each gene being an instance of a particular "allele" (e.g., 0 or 1). Recombination operators are applied to this binary string so as to preserve critical information. A GA commences with a population of typically randomly generated chromosomes. The selection operator chooses which chromosomes in the population will be allowed to reproduce. The chromosomes are allocated reproductive opportunities in a way that chromosomes which represent a better solution to the target population are given a higher chance to "reproduce" than those chromosomes which have achieved lower fitness evaluations. It also decides how many offspring each chromosome can have, with fitter chromosomes having a greater chance to produce more offspring. The fitness of a solution is typically defined with respect to the current population.

Following selection the crossover operator exchanges subparts of two chromosomes, normally with a probability rate typically between 85% and 90%, this can be

compared to biological recombination between two single-chromosome organisms, and may be viewed as sexual recombination found in nature. The crossover operator randomly chooses a locus and exchanges the parts of the strings before and after that locus between two chromosomes to create two offspring. For example, the strings *10000100* and 11111111 could be crossed over after the third locus in each to produce the two offspring *100*11111 and 111*00100*. The mutation operator changes randomly the values of a location on the chromosome. It randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001) [16]. The order in which genes are arrayed in the chromosome can be rearranged by the inversion operator, which reverses the order of a contiguous section the chromosome [17]. Most methods called "GAs" have the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring.

3 Understanding Genetic Algorithms

Holland puts forward the *schema theorem* to explain how a GA can result in a complex and robust search by implicitly sampling hyperplane partitions of a search space [5]. A Schema is a collection of gene values which may be represented (in a binary coding) by a string of characters contained within the alphabet 0, 1, *. A chromosome contains a particular schema if it matches that schemata, with the "*" symbol matching anything. For example, the chromosome "1010" contains many schemata including "10**", "*0*0", "**1*" and "101*". The order of a schema is the number of non-*'s it contains (2,2,1,3 respectively in the above examples). The defining length of a schema is calculated by the distance between the outermost non-* symbol (2,3,1,3 respectively).

The schema theorem explains the power of a GA in terms of how schemata are processed. Members of the population are individually given the chance to reproduce, and create offspring. The number of chances a individual receives is directly related to that individuals fitness, therefore fitter individuals contribute more of their genes to the next generation. It is assumed that a creatures high fitness is due to the fact that it contains good schemata. By passing more good schemata to the next generation the liklihood of finding a better solution is increased. Holland showed that the optimum way to explore the search space is to allocate reproductive opportunities to creatures in proportion to their fitness relative to the other creatures in the population. By doing this good schemata receive an exponentially increasing number of chances to reproduce in successive generations. Holland also showed that as each creature contains many different schemata, the number of schemata effectively being processed in each generation is of the order n^3 (n being the population size), this property is called *implicit parallelism* [2].

Goldberg [6] argues that the power of a GA lies in being able to find good *building blocks*. These building blocks are schemata of short defining length consisting of bits that work well together, and tend to improve performance when incorporated into a creature. A successful coding scheme encourages building blocks to form by ensuring that, (I) related genes are close together on the chromosome, while (II) there is little interaction between genes. *Interaction* (also called *epistasis*) between genes means that the contribution of a gene to the fitness depends on the fitness of other genes in the chromosome. GA researchers use the term epistasis to refer to any kind of strong interaction among genes. There is always some epistasis between genes in multi-modal fitness functions. Multi-modal functions are important in GA research because unimodal functions can be solved using simpler methods. If (I) and (II) above are observed them GA's will be as effective as predicted by the schema theorem. However genes may be related in ways which do not allow all closely related to be situated close together in a one-dimensional string, also the programmer may not know the exact relationship between the genes which, may prevent the coding scheme form being implemented as successfully as possible. The condition that there is little interaction between genes is a precondition for the condition that related genes are close together on the chromosome. Suppose the contribution to overall fitness of each gene were independent of all other genes, then it may be possible to solve the problem by hill-climbing on each gene in turn. But generally this is not possible. If the programmer can ensure that each gene interacts with a small number of other genes and these can be placed on the chromosome, then both (I) and (II) can be adhered to. If on the other hand there is a lot of interaction between genes then neither condition can be met. All this implies that when designing the code schemes we should try to conform with Goldberg's building block hypothesis so as to ensure that the GA will perform as well as possible.

4 Operators

4.1 Operator overview

Reproduction combined with Crossover provide GA's with the bulk of their processing power. Mutation on the

other hand plays a secondary role in the operation of genetic algorithms. It is needed because, "even though reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and loose some potentially useful genetic material (1's or 0's at particular locations). In artificial genetic systems, the mutation operator protects against such an irrecoverable loss. By itself, mutation is a random walk through the string space. When used sparingly with reproduction and crossover, it is an insurance policy against premature loss of important notions". Goldberg notes that the frequency of mutation to obtain good results in empirical GA studies is "on the order of one mutation per thousand bit (position) transfers".

4.2 Crossover Operator

The traditional GA uses 1-point crossover, where two mating chromosomes are each cut once at corresponding points, and the segments following the cuts are exchanged [1]. Crossover is the predominant operation in genetic algorithm work and is performed with a high probability (say, 85% to 90%) [16]. There are however many different crossover algorithms which have been devised, these often include more than one cut point. The problem with adding additional crossover points is that it will increase the likely hood of building blocks being disrupted. This being said, an advantage of having additional building blocks is that the problem space may be searched more thoroughly.

In a 2-point crossover and multi-point crossover in general, chromosomes are seen as loops formed by joining the ends together, rather than as linear strings. To crossover a section from one loop to another requires two cut points to be chosen. When viewed as a loop 1-point crossover can be seen as 2-point crossover with one of the cut points fixed at the start of the string. Thus 2-point crossover performs the same task as 1-point crossover, but it is more general. A chromosome viewed as a loop is able to contain more building blocks as they are able to wrap around at the end of the string. Researchers now agree that 2-point crossover produces better results than 1-point crossover [2]. However, if a strict interpretation of the schema theorem is imposed then operators which use many crossover points should be avoided because they can cause extreme disruption to schemata [5]. A completely different crossover from the 1-point crossover is the *Uniform Crossover*. Each gene in the offspring is created by copying the corresponding gene from one of the parents, chosen according to a randomly generated *crossover mask*. Where a 1 exists in the crossover mask, the gene is copied from the first parent, and where a 0 exists in the mask the corresponding gene is copied from the second parent. The same process is carried out again to produce a second offspring.

The choice of which crossover operator is the most efficient is still being debated. Syswerda [18] argues that the uniform crossover is superior, claiming that "under uniform crossover, schemata of a particular order are equally likely to be disrupted, irrespective of their defining lengths". If a 2-point crossover is used, it is the defining length of the schemata, not its order, which determines its likelihood of disruption. What this means is that under uniform crossover, although short defining lengths run the increased risk of being disrupted, longer defining length schemata are comparatively less likely to be disrupted [1]. Syswerda claims that therefore the total amount of schemata distribution is lower. Uniform crossover has the advantage that the ordering of genes is irrelevant. This implies that ordering operators such as inversion are not required, as there is no need to worry about positioning genes so as to promote building blocks. However Spears & DeJong [14] are critical of multi-point and uniform crossover. They adhere to the traditional analysis, which illustrates that 1- and 2- point crossover are optimal. They argue that 2-point will perform poorly when the population has to a large extent converged, due to reduced *crossover productivity* (the ability of the crossover operator to produce offspring which sample different points in the search space). Where two chromosomes are similar, the sections exchanged by 2-point crossover are likely to be identical, which will lead to offspring identical to their parents. There is a reduced chance of this happening with uniform crossover. Spears & DeJong then describe a new 2-point crossover operator such that if identical offspring are produced, two new cross points will be chosen. This operator was then tested and proved marginally better than the uniform crossover on a test problem. It is also stated that Booker [3] introduced a *reduced surrogate crossover* to achieve the same effect. Spears & DeJong concluded in later papers that modified 2-point crossover is better for large populations, but the increased disruption of uniform crossover is more beneficial for problems with small population sizes [1].

The original analysis of GA's present combination to be the primary mechanism of crossover. However, although good solutions can be found they are not always locally optimal. A popular technique is to locally optimise each crossover solution before adding it to the population. Within these "hybrid" operators, crossover can be viewed as a means of restarting the local optimiser. However, if crossover does little more than combine random parts of two parent solutions, the performance of the resulting hybrid operator may not be significantly different from random restart of the local optimiser. The design of the crossover affects the efficiency and effectiveness of hy-

brid operators.

The traditional design focus for crossover operators has been combination - random parts (schemata) from both parents are crossed into an offspring solution. This design basis is a legacy of the original Schemata Theorem. The focus is on implicit parallelism and the idea of sampling of schemata, particularly low-order schemata (building blocks). But the original analysis of crossover was based on a standard representation, which uses the viewpoint of a single parent, and is pre-occupied with disruption. Non-standard representations and hybrid operators were never intended extensions of the original design focus of combination. Research carried out by Chen & Smith [4], developed a new Schema Theorem, "the analysis of crossover is representation independent, and uses the view point of both parents and ignores disruption. The new Schema Theorem suggests that common schemata are important. Instead of combining random schemata a common schemata of two parents are used as a foundation on which a new offspring is built". Expressly a new two step process is defined for crossover. First, "preserve the maximal common schema of two parents", and second, "complete the solution within a constructive heuristic". The scope of Chen & Smiths design framework extends to non-standard representations and hybrid operators. The results of this research have shown that the defining attribute of a genetic operator should be crossover. A well designed crossover operator can improve the performance of a local optimiser. The authors believe that this re-establishes the role for crossover in hybrid operators, and that the overall search method can again be called a genetic algorithm.

4.3 Selection Operator

For each fitness problem a fitness function must be devised. Given a particular chromosome, the fitness function returns a numeric "fitness" value which is supposed to be proportional to the ability of the individual creature which that chromosome represents. For functional optimisation problems the fitness function should measure the value of the function. However this is not as obvious for many problems i.e. combinatorial optimisation. Along with the coding scheme adopted, the fitness function is the most crucial aspect of a GA. Ideally what is required is a smooth, regular fitness function so that the chromosomes with reasonable fitness are close (in parameter space) to chromosomes with slightly better fitness. However for many it is not possible to construct ideal fitness functions, otherwise hill-climbing algorithms could be used. This aside fitness functions need to be constructed which do not have too many local maxima or very isolated global maxima, if the GA is to perform well. The general rule in constructing a fitness function is that it should reflect in some "true" way the value of the chromosome. However the "true" value of a chromosome may not always be a useful for guiding genetic search. With combinatorial optimisation problems, where there are many constraints, most points in the search space often represent invalid chromosomes, hence having a zero "true" value.

Take a timetabling problem for example, to construct a school timetable a number of classes need to be given a number of lessons. There are a finite number of rooms and lecturers available. Most allocations of classes and lecturers to rooms will violate constraints such as a room being occupied by two classes at once, a class or lecturer being in two places at once, or a class not being timetabled for all the lessons it is supposed to receive. For a GA to operate successfully a fitness function needs to be created where the fitness of an individual chromosome is seen in terms of how it is leading us towards valid chromosomes. This is in fact a sort of catch-22 situation, it should be known where the valid chromosomes are to ensure that the nearby points can also be given a good fitness values, and far away points given poor fitness values, but if it is not known where the valid chromosomes are this cannot be achieved. It has been suggested by Cramer [12] that if the goal of the problem is all or nothing, better results can be obtained if meaningful sub-goals are invented. These sub-goals should then be rewarded. For example with the timetabling problem a reward could be given for the classes which had its lessons allocated validly. Another approach which can been taken in this situation is to use a penalty function, which represents how poor the chromosome is and construct the fitness as (constant minus penalty) [6]. A number of guidelines for constructing penalty functions have been produced, these include fitness functions whereby penalties which represent the amount by which the constraints are violated are considered better than those which are based on the number of constraints which are violated. Good penalty functions can be constructed from the expected completion cost. In other words, given another invalid chromosome, how much will it"cost" to turn it into a valid one? [2].

If the fitness function is excessively slow or complex to evaluate a technique known as *Approximate function evaluation* may be used. If a much faster function can be devised which approximately gives the value of the"correct" fitness function, the GA may find a fitter chromosome in a given amount of CPU time than when using the "correct" fitness function. For example if the simplified function is ten times faster, ten times as many function evaluations can be performed in the same amount of time. An approximate evaluation of ten points in the search space is normally better than an exact evaluation of one [2]. A GA is considered

robust enough to be able to converge in spite of the noise introduced by the approximation. Approximate fitness functions have been used in instances where the fitness function is stochastic [6].

4.4 Mutation

Traditionally mutation is seen as a"background" operator responsible for re-introducing mistakenly lost gene values (alleles), preventing genetic drift, and providing a small element of random search in the area of the population when it is has largely converged. It is normally held that crossover is the main force leading to a methodical search of the problem space. However, examples in nature show that asexual reproduction can evolve sophisticated creatures without crossover [1]. Tate and Smith [19] argue that the optimal mutation rates depend strongly on the choice of encoding, and that problems requiring non-binary encoding may benefit from mutation rates much higher than those generally used with binary encodings. The authors introduce the idea of the expected allele coverage of a population and discuss its role in guiding the choice of mutation rate and population size. Many biologists see mutation as the main source of raw material for evolutionary changes. This theory moves away from the traditional view that low mutation rates are used in GA's as they tend to lead to an efficient search of the solution, and that high mutation rates result in diffusion of search effort and premature extinction of favourable schemata in a population. The article begins by outlining the traditional role of mutation as formulated by Goldberg (discussed above). Following this the authors show that this characterisation is sensitive to the complexity of the encoding, they introduce the concept of *expected allele coverage* as a measure of the degree to which exploratory mutation is or is not necessary. They then argue that certain classes of optimisation problems require encodings with low expected allele coverage, and thus low mutation implementations are not always ideal.

Two general comments on mutation are criticised by the article. The first is Goldberg's view that mutation is nothing more than a random walk through string space. The authors see this as being valid only when mutation is applied *uniformly* over the population, without regard to fitness. Selection pressures act on individuals without regard to whether their phenotypes are the result of breeding or mutation. The second traditional view is that mutation, if not used sparingly, is destructive. But in a steady state GA, where the majority of the population persists unchanged from one generation to the next, this, the authors believe may not be a problem. Also the danger of eliminating desirable schemata is reduced significantly if you allow the possibility that both a mutant encoding and the individual which gave rise to it might survive into future generations. This allowed the possibility of mutation as a local search mechanism in the neighbourhood of highly fit solutions, without the danger of eliminating the highly fit solutions during the process.

Mutation, which is primarily useful for retrieving valuable schemata implies the belief that the mutation rate ought to be set so that the rate at which highly fit schemata are accidentally deleted from the population is roughly equal to the rate at which mutations introduce new desirable schemata. Implied in this view is that it is assumed there is no need to find highly fit schemata that are not present in the initial population, or that cannot be generated from that population by repeated breeding operations. Put another way, this view assumes that nearly all the useful gene alleles are present in the initial population. The authors have concluded that for simple string encoded GA's low mutation rates are sufficient. This is because of the high expected allele coverage exhibited by random populations of binary strings. More complex encodings on the other hand exhibit lower expected coverage. This can be adjusted for, to some degree, using non-random initial populations, but a more efficient way of coping with lower coverage is to use higher mutation rates. In order to achieve these higher mutation rates without degenerating into simple random search, it is useful to close off mutation and breeding into separate, parallel processes. The relative frequency of breeding and mutation can be controlled exactly in this manner, without allowing the researcher to search randomly for highly-fit but previously unseen allele values through the breeding of unmutated offspring. The authors also point out that it is useful in such cases to use steady-state GA implementation in which highly fit solutions can persist from one generation to the next , even while generating mutants, and poor solutions are culled from the population.

4.5 Inversion and Reordering

The order of genes on a chromosome is critical for the building block hypothesis to work effectively. A number of techniques for reordering the positions of genes in a chromosome during run time have been suggested. One technique *inversion* works by reversing the order of genes between two randomly chosen positions within the chromosome. If these techniques are being used then genes must carry with them a sort of identification code so that they may be correctly identified irrespective of their position within the chromosome. It is pointed out that reordering greatly expands the search space. By doing this not only is the GA trying to locate a good set of gene values, it is simultaneously trying to discover good gene orderings too. This is a

far more difficult problem to solve. The computational time spent looking for better gene ordering may mean time taken away from finding good gene values. Beasley et al. [2] describe inversion as mechanism by which the arrangement of the chromosome(s) may evolve, this is known as *karyotypic evolution* in nature. Generally inversion is implemented by reversing a random segment of the chromosome. However bits need to have a position independent decoding before moving bits around to improve linkage. Without position independent coding reversing segments of bits is similar to large-scale mutation. Position independent coding requires that each bit be tagged in some way. Consider the following example, `((9 0)(6 0)(2 1)(7 1)(5 1)(8 1)(3 0)(1 0)(4 0))` the first number is the bit tag which indexes the bit and the second represents the bit value. The linkage can now be changed by moving the tag-bit pairs around, but the string remains the same when decoded `(010010110)`.

The type of crossover in operation can have an influence on an inversion operator, sometimes negatively. For example, the purpose of reordering is an attempt to find gene orderings, which have better evolutionary potential. However, reordering does nothing to lower the epistasis, therefore cannot help with the other requirement of the building block hypothesis. Also it cannot help if the relationships among the genes do not allow a simple linear ordering. Therefore if a uniform crossover is in operation then gene order is irrelevant, so as Syswerda [18] argues in this case there is no need for inversion [1].

5 Problems With GA's

There are several open problems associated with GA's. Many of these problems are of a technical nature (e.g. questions about genetic operators and their representations) and others are more general and relate to many areas of artificial life. This literature review is concerned with the questions, which arise about genetic operators and their representations. Specifically to GA's is the central question of representation. For any problem domain, the choice of which features to represent on the genotype and how to represent them is crucial to the GA's performance. The choice of features which compromise the genotype is a decision which has to be decided at the design stage and cannot be automated. GA's typically use low level primitives such as bits, which can be worlds away from the natural representation of environmental states and control parameters. Because of this the representation problem is especially important for GA's [17].

5.1 Representation Problem

Representation is a major issue for GA's because GA's directly manipulate a coded representation of the problem, and because the representation scheme can limit the window through which a system observes its world. Although traditional GA's operating on fixed-length character strings are capable of solving a large number of problems, there are a number of areas where they may not be suitable [11].

Within the field of GA's efforts towards getting programs to learn to solve problems without being explicitly programmed have focused on providing a greater level of flexibility by using increasingly more complex representations. Although the representation problem has been recognised there have been very few innovative representations, notable exceptions being messy GA's. Messy algorithms were introduced by Goldberg, Korb and Deb in [7], which processed populations of variable length character strings. They solve problems by combining relatively short, well tested sub-strings that deal with part of a problem to form longer more complex strings that will deal with part of a problem to form longer, more complex strings that will deal with more complex aspects of the problem. Also domain-specific structures that are more complex than character strings have been devised and applied to many combinatorial optimisation problems such as the Travelling Salesman Problem (TSP). For the TSP the crossover operation has been modified in an application-specific way to either; first, maintain"syntactic legality while preserving the building blocks relevant to the particular application". and second, "to repair syntactic illegality while preserving the building blocks relevant to the application, or third to compensate for syntactic illegality in some manner appropriate to the application" [11].

5.2 Choice of Operators

Representation issues tend to address the question of how to engineer GA's. Related to representation issues is the choice of genetic operators for introducing variation into a population. One reason that binary linearly ordered representations are frequently used is that standard Mutation and Crossover operators can be applied in a problem independent way. Other operators have been experimented with for various optimisation problems, but so far no new general-purpose operators have been widely adopted since the advent of GA's. The opposite in fact is true, the Inversion operator, which was included originally for theoretical reasons, has to a large extent been abandoned. It has been suggested that perhaps that more research should be carried out on this operator [17].

New versions of the Crossover operator have also been developed such as the Uniform Crossover. This can reduce the inherent bias in standard crossover of breaking up correlated genes that are widely separated on the chromosome, known as"positional bias". These approaches are viewed as promising in some cases, especially since the strong positional dependence of most current representations is an artifact introduced by GA's. "In natural genetic systems, one gene (approximately) codes for the one protein regardless of where it is located, although the expression of a gene (when the protein is synthesised) is indirectly controlled by its location". In spite of this, most current GA implementations use a simple binary alphabet linearly ordered along a haploid string. Researchers interested in engineering applications have believed that the use of "high-cordiality alphabets" including real numbers as alleles should be advocated [17].

6 The Fitness Landscape

6.1 Bit strings and Hamming distance

To describe fitness landscapes, a notion of a distance between genotypes is needed. Genotypes are codings, and different codings can cause different distance measures. Another point is that often there are more than one distance measure, or metric, which can be defined for one and the same coding. Usually, a coding in the form of bit strings is used. The Hamming distance between two bit strings is defined as the number of corresponding positions in these bit strings where the bits have a different value. So, the distance between 010 and 100 is two, because the first and second positions have different values. A normalised Hamming distance can be defined by dividing the Hamming distance between two bit strings by the length of these bit strings. By adopting this approach, the distance measure is independent of the length of the bit strings. A normalised Hamming distance of 0.5, for example, means that half the bits of two bits string have a different value [9].

6.2 The Genotype Space

If the possible solutions for a given problem are encoded by some form of genotype, then the problem space can also be represented by a genotype space. A genotype space is the space in which each point represents one genotype and is next to all other points that have a distance of one from this point (according to some appropriate metric i.e. the Hamming distance). All the points at distance one are called the neighbours of this first point, and together they form a neighbourhood, i.e. consider as genotypes bit strings of length 3. The total number of bit strings of this length is 2^3 = 8. With the Hamming distance as metric, every bit string of length three has exactly three neighbours, namely those bit strings that differ in one of the three bits. Every point in the space represents one genotype and has exactly three neighbours, each of which differs in the value of one bit [9].

6.3 The Fitness Landscape

Shortly after the first mathematical models of Darwinian evolution were developed, Seawall Wright (1932) recognised a deep property of population genetic dynamics, that is when fitness interactions between genes, the genetic composition of a population can evolve into multiple domains of attraction. The specific fitness interaction is known as *epistasis*, where the effect on fitness from altering one gene depends on the allelic state of other genes. Epistasis makes it possible for the population to evolve toward different combinations of alleles, depending on its initial genetic combination. Thus Wright discovered a conceptual link between a microscopic property of organisms (fitness interactions between genes) and a macroscopic property of evolutionary dynamics (multiple population attractors in the space of genotypes. Wright illustrated this by using the metaphor of a landscape of multiple peaks, in which a population would evolve by moving up hill until it reached its local fitness peak. This visualisation of the *adaptive landscape* is the term used to describe multiple domains of attraction in evolutionary dynamics. Wright looked specifically at how populations could get away from local fitness peaks to higher ones through stochastic fluctuations in small population subdivisions. This was one of the first conceptions of stochastic processes for the optimisation of multi-modal functions [13].

The idea of a fitness landscape is used as a framework for thinking about evolution. Biological organisms can be characterised by their genotype, which is the genetic "encoding" of the organism, or their phenotype, which is the actual form and behaviour of the organism. A fitness value can be assigned to each phenotype, which denotes its ability to survive and reproduce. Evolution can be viewed as a process that searches, by means of genetic operators like crossover and mutation, a fitness landscape of possible genotypes, looking for genotypes that encode highly fit phenotypes. Put another way, evolution searches for solutions, encoded in genotypes, for the problem of finding fit organisms that are capable of reproduction. Every genotype will have a relative fitness assigned to it. This is determined by a fitness function. The fitness landscape is then constructed by assigning the fitness values of the genotypes to

the corresponding points in the genotype space. To visualise this picture each point in the genotype space being given a "height" according to its fitness. From this way a "mountainous" landscape is formed, where the highest peaks designate the best solutions. A local optimum, or peak, in such a landscape is defined as a point that has a higher fitness than all its neighbours [10].

This landscape paradigm can be used to describe search in general. Given an optimisation problem, the set of possible solutions to this problem can be encoded using strings of a (normally) fixed length formed from some finite size alphabet. This encoding generates a representation space, which is a high dimensional space of all possible strings of a particular length, each of which encodes a possible solution to the problem. In addition, there is a neighbourhood relation that defines which points in the representation space are associated. This relation usually depends on the specific search operator i.e. mutation, or combination of search operators, i.e. crossover and mutation, that are used to search the space. That is to say, points that are reachable from each other by one application of the search operator are connected. Finally, there is a fitness function, which assigns a fitness value to each possible word, or point in the landscape. This fitness value can be seen as a measure of how good a solution, represented by that point in the landscape, is to the given problem. Therefore we can conclude that a fitness landscape is defined by three things: A representation space (i.e. all the possible strings in the encoding). A neighbourhood relation denoting, which points in the representation space are neighbours. And finally a fitness function that assigns a fitness value to each point in the space [10].

These give rise to the mountainous landscape, with the fitness of each point being represented by its height. Optimisation, i.e. finding good solutions to a problem, is now a search of this landscape, looking for the highest peaks, which mean the best solutions. Different landscapes differ in their ruggedness. A landscape with a few local peaks and small (average) fitness differences between neighbouring points is called smooth. A landscape with a lot of local peaks and large fitness differences is called rugged. The more rugged a landscape is, the harder it will be to search on it for the best solutions, and the less information there will be about the fitness of distant points in the landscape. The fitness landscape metaphor can help in understanding evolutionary processes or search processes in general. Furthermore, the above definition of a fitness landscape lends itself to mathematical analysis, also the structure of a landscape can reflect how easy or difficult it is for a search algorithm to find good solutions.

To summarise, the structure of a landscape incorporates many things, like the number of neighbours each point in the genotype space has, the number of peaks, the "steepness" of the hillsides, the relative height of the peaks, etc. A landscape where the average difference in fitness between neighbouring points is relatively small, is called smooth. On such a landscape it will be easy to find good optima as local information about the landscape can be used effectively to direct the search. A landscape with a relatively large average fitness difference between neighbours, is called rugged. On such a landscape it will be difficult to find good optima, in other words local information becomes less valuable. So, the global structure of a landscape can range from very smooth to very rugged. The structure of a fitness landscape depends on the underlying problem. But a theory about population flow should be independent of that. It would therefore be convenient to have a problem independent fitness landscape [9].

7 NK Model

7.1 Description of the NK Fitness Model

> "We need a real theory relating the structure of rugged multipeaked fitness landscapes to the flow of a population upon those landscapes. We do not yet have such a theory."
>
> —Stuart A. Kauffman

Stuart Kauffman devised the"NK fitness Landscape" model to explore the way that epistasis controls the "ruggedness" of a adaptive landscape. He wanted to specify a family of fitness functions whose ruggedness could be 'turned' by a single parameter. This was achieved by building up multiple 'atoms' of maximal epistasis. The NK model is a stochastic method for generating a fitness function F:$\{0,1\}^N \rightarrow \mathcal{R}^+$ on binary strings $x \in \{0,1\}^N$, where the genotype x consists of N loci, with two possible alleles at each locus x_i. It has two basic components: a structure for gene interactions, and a way this structure is used to generate a fitness function for all possible genotypes. The gene interaction structure is created as follows: the genotype's fitness is the average of N fitness components contributed by each locus. Each gene's fitness component F_i is determined by its own allele, x_i, and also the alleles at K other epistatic loci (therefore K must fall between 0 and N - 1). These K other loci could be chosen in any number of ways from the N loci in the genotype. Kauffman investigated two different alternatives: *adjacent neighbourhoods*, where the K genes nearest to locus i on the chromosome are chosen; and *random neighbourhoods*, where these K other

loci are chosen randomly on the chromosome. In the adjacent neighbourhood model, the chromosome is taken to have periodic boundaries, so that the neighbourhood wraps around the other end when it is near the terminus [13].

Epistasis is implemented through a"House of Cards" model of fitness effects, in other words, whenever an allele is changed at one locus, all of the fitness components with which the locus interacts are changed, without any correlation to their previous values. Thus a mutation in any of the genes affecting a particular fitness component is like pulling a card out of a house of cards - it tumbles down and must be rebuilt from scratch, with no information passed on from the previous value. Kauffman implemented this by generating, for each fitness component, a table of 2^{K+1} numbers for each allelic combination for the K + 1 loci determining that fitness component. These numbers are independently sampled from a uniform distribution on (0,1). The consequence of this individual resampling of fitness components is that the fitness function develops conflicting constraints: a mutation at one gene may improve its own fitness component, but decreases the fitness component of another gene with which it interacts. Also, if the allele at another interacting locus changes, an allele that had been optimal, given the alleles at the other loci, may no longer be optimal. Therefore, epistatis interactions provide "frustration" in trying to optimise all genes simultaneously[13].

The NKmodel was introduced by Kauffman to have a problemindependent model for constructing fitness landscapes that can gradually be tuned from smooth to rugged. The main parameters of the model are N, the number of genes in the genotype, i.e. the length of the strings that form the points in the landscape, and K, the number of other genes that epistatically influence a particular gene (i.e., the fitness contribution of each gene is determined by the gene itself plus K other genes) [9]. K sets the level of epistasis by determining the dependence the partial fitness of a gene at location n has on the genes in a neighbourhood of K other locations. The neighbourhood may be at the K locations nearest to n in the genotype or a set of K locations randomly picked from anywhere on the genotype. Following this a series of N lookup tables are then generated, one for each gene location in the genotype. Each table has 2^{K+1} random entries in the interval (0,1). The fitness, F_{NK}, of a particular genotype is calculated by the function:

$$F_{NK} = \frac{1}{N} \sum_{n=1}^{N} f(x)$$

where the partial fitness f(n) is obtained from the nth lookup table using the values of the genes in location n and its neighbourhood as the lookup key [15]. Below is an example to illustrated the calculation of $f(n)$ with N=8, K=2. In this example n = `011`, when the table is referred to $f(n)$ is shown to be `0.432809`.

`Genotype:`

`Neighbourhood of n`

1	0	0	0	1	1	0	0

nth Lookup Table

0	0	0	0.724367
0	0	1	0.123989
0	1	0	0.987432
0	1	1	0.432809
1	0	0	0.987234
1	1	0	0.349566
1	1	0	0.274095
1	1	1	0.521926

Many experiments have used Kauffman's NK model, one such example is an experiment by Giles Mayley [15]. Mayley choose the Kauffman's NK fitness Model because a landscape was desired that provided many local optima for search such that evolution could adopt a genetically fixed solution at a lower maximum than one found by learning is the costs of learning were too high, the NK model provided an arbitrary fitness landscape of specified ruggedness. Therefore the generalised NK model can be applied to the representation problem in evolutionary computation, i.e. how to represent the objects in the search space so that genetic operators can have a good chance of producing fitter variants when acting on the representation [13]. The NK model can be used as an abstraction for the way representations produce epistatic interactions between genes.

8 Conclusion

8.1 Summary

GA's are search algorithms based on the mechanics of natural selection and natural genetics. A GA begins with a randomly generated population. The selection operator selects which chromosomes in the population will reproduce.

The chromosomes are allocated reproductive opportunities in a way that chromosomes which represent a better solution to the target population are given a higher chance to reproduce than those with a lower fitness level. The fitness of a solution is typically defined with respect to the current population. Following selection the crossover operator exchanges subparts of two chromosomes, typically with a probability between 85% and 90%. Mutation may then be applied normally with a probability of 1 in a 1000. Inversion and other reordering operators may also be applied depending on the design of the the GA and the problem to be solved. The fitness landscape is constructed by assigning the fitness values of the genotypes to the corresponding points in the genotype space, each genotype is given a"height" according to its fitness. The global structure of a landscape can range from very smooth to very rugged, this depends on the underlying problem. However, to develop a theory about population flow a problem independent fitness landscape would be convenient. The NK model is designed to explore the way the epistasis controls the "ruggedness" of an adaptive landscape, it was introduced by Kauffman to have a problem-independent model for constructing fitness landscapes that can gradually be turned from smooth to rugged. The generalised NK model can be applied to the representation problem in evolutionary computing, that is how to represent objects in the search space so that the genetic operators can have a good chance of producing fitter variants when acting in the representation.

8.2 Future Work

From the above it is clear that much research has been conducted on GA's and their operators. However, there has been an increase in the numbers of papers being published on Genetic Algorithms in recent years, indicating that much work still remains outstanding. The representation problem is one of the largest problems encountered when designing a GA. Related to this is the choice of genetic operators for introducing variation into a population. Although operators have been experimented with in optimisation problems, no new general purpose operators have been widely adopted since GA's were created. The opposite is true, the inversion operator which was included in the original proposal for theoretical reasons has largely been abandoned. This operator I feel, may require more research, possibly using Kauffman's NK model to analyse the performance of the inversion operator and to see if a more efficient search of the search space can be implemented.

References

[1] D. Beasley, D. R. Bull, and R. R. Martin. Genetic algorithms, part 2 research topics. Technical report, University of Purdue, 1993.

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

[3] L. Booker. Improving search in genetic algorithms. In D. L., editor, *Genetic Algorithms and Simulated Annealing*, pages 100–107. Morgan Kaufmann, 1987.

[4] S. Chen and S. F. Smith. Putting the "genetics" back into genetic algorithms (reconsider the role of crossover in hybrid operators). Technical report, The Robotic Institute, Carnegie Mellon University, 1999.

[5] W. Darrell. A genetic algorithm tutorial. Technical Report CS-93-103, Colorado State University, 1993.

[6] D. E. Goldberg. *Genetic Algorithms in Search Optimization, and Machine Learning*. Addison-Wesley, 1989.

[7] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1990.

[8] H. J. H. *Adaptation in Natural and Artificial Systems*. The University of Michigan, 1975.

[9] W. Hordijk. *Population flow on fitness landscapes*. PhD thesis, University of Rotterdam 1994, 1994.

[10] W. Hordijk. A measure of landscapes. Technical report, Santa Fe Institute, 1995.

[11] K. Joh. *Genetic Programming*. The MIT Press, 1992.

[12] C. N. L. A representation for the adaptive generation of simple sequential programs. In J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 183–187. Lawrence Erlbaum Associates, 1985.

[13] A. Lee. Nk fitness landscapes. Technical report, Hawai'i Institute of Geophysics and Planetology, University of Hawai'i at Manoa, Honolulu, HI USA., 1996.

[14] S. W. M. and K. DeJong. An analysis of multi-point crossover. In *Foundations of Genetic Algorithms*, pages 301–315. Morgan Kaufmann, 1991.

[15] G. Mayley. The evolutionary cost of learning. Technical report, School of Cognitive and Computer Sciences, University of Sussex, 1996.

[16] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, USA, 1996.

[17] F. S. Mitchelle Melanie. Genetic algorithms and artificial life 93-11-072. Technical report, Santa Fe Institute, 1993.

[18] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on article Genetic Algorithms*, pages 2–9. Morgan Kauffmann, 1989.

[19] D. Tate, M. David, and A. E. Smith. Expected allele coverage and the role of mutation in genetic algorithms. Technical report, Dept. of Industrial Engineering, University of Pittsburg, 1993.