Genetic algorithms overview

Franco Busetti

1 Introduction and background

Note: Terminology will be developed within the text by means of *italics*.

Genetic algorithms (*GAs*) are adaptive methods which may be used to solve search and optimisation problems. They are based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest. By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, if they have been suitably encoded. The basic principles of GAs were first laid down rigorously by Holland [1].

GAs work with a *population of "individuals*", each representing a possible solution to a given problem. Each individual is assigned a *"fitness score*" according to how good a solution to the problem it is. The highly-fit individuals are given opportunities to "*reproduce*", by "*cross breeding*" with other individuals in the population. This produces new individuals as "*offspring*", which share some features taken from each "*parent*". The least fit members of the population are less likely to get selected for reproduction, and so "*die out*".

A whole new population of possible solutions is thus produced by selecting the best individuals from the current "generation", and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the population. By favouring the mating of the more fit individuals, the most promising areas of the search space are explored. If the GA has been designed well, the population will *converge* to an optimal solution to the problem.

2 The method

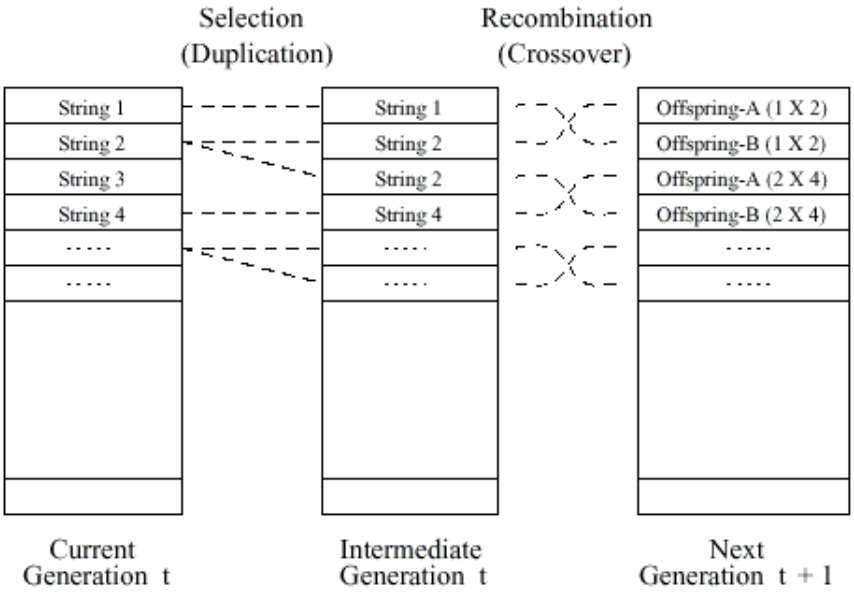
2.1 Overview

The *evaluation function*, or *objective function*, provides a measure of performance with respect to a particular set of parameters. *The fitness function transforms that measure of performance into an allocation of reproductive opportunities*. The evaluation of a string representing a set of parameters is independent of the evaluation of any other string. The fitness of that string, however, is always defined *with respect to other members of the current population*. In the genetic algorithm, fitness is defined by: f_i / f_A where f_i is the evaluation associated with string i and f_A is the average evaluation of all the strings in the population.

Fitness can also be assigned based on a string's *rank* in the population or by sampling methods, such as *tournament selection*. The execution of the genetic algorithm is a two-stage process. It starts with the

current population. Selection is applied to the current population to create an intermediate population. Then *recombination* and *mutation* are applied to the intermediate population to create the next population. The process of going from the current population to the next population constitutes one *generation* in the execution of a genetic algorithm.

The standard GA can be represented as follows:



In the first generation the current population is also the initial population. After calculating f_i / f_A for all the strings in the current population, selection is carried out. The probability that strings in the current population are copied (i.e. duplicated) and placed in the intermediate generation is in proportion to their fitness.

Individuals are chosen using "*stochastic sampling with replacement*" to fill the intermediate population. A selection process that will more closely match the expected fitness values *is "remainder stochastic sampling*." For each string i where f_i/f_A is greater than 1.0, the integer portion of this number indicates how many copies of that string are directly placed in the intermediate population. All strings (including those with fi/f_A less than 1.0) then place additional copies in the intermediate population with a probability corresponding to the fractional portion of fi/f_A. For example, a string with fi/f_A = 1:36 places 1 copy in the intermediate population, and then receives a 0:36 chance of placing a second copy. A string with a fitness of fi/f_A = 0:54 has a 0:54 chance of placing one string in the intermediate population. Remainder stochastic sampling is most efficiently implemented using a method known as *stochastic universal sampling*. Assume that the population is laid out in random order as in a pie graph, where each individual is assigned space on the pie graph in proportion to fitness. An outer roulette wheel is placed around the pie with N equally-spaced pointers. A single spin of the roulette wheel will now simultaneously pick all N members of the intermediate population.

After selection has been carried out the construction of the intermediate population is complete and *recombination* can occur. This can be viewed as creating the next population from the intermediate population. *Crossover* is applied to randomly paired strings with a probability denoted p_c. (The population should already be sufficiently shuffled by the random selection process.) Pick a pair of strings. With probability p_c "*recombine*" these strings to form two new strings that are inserted into the next population.

Consider the following binary string: 1101001100101101. The string would represent a possible solution to some parameter optimization problem. New sample points in the space are generated by recombining two parent strings. Consider this string 1101001100101101 and another binary string, yxyyxyxxyyyxyxxy, in which the values 0 and 1 are denoted by x and y. Using a single randomly-chosen recombination point, 1-point crossover occurs as follows:

11010 \/ 01100101101
yxyyx /\ yxxyyyxyxxy

Swapping the fragments between the two parents produces the following offspring:

11010yxxyyyxyxxy and yxyyx01100101101

After recombination, we can apply a *mutation operator*. For each bit in the population, mutate with some low probability p_m. Typically the mutation rate is applied with 0.1%-1% probability. After the process of selection, recombination and mutation is complete, the next population can be evaluated. The process of valuation, selection, recombination and mutation forms one *generation* in the execution of a genetic algorithm.

2.2 Coding

Before a GA can be run, a suitable *coding* (or representation) for the problem must be devised. We also require a fitness function, which assigns a figure of merit to each coded solution. During the run, parents must be selected for reproduction, and recombined to generate offspring.

It is assumed that a potential solution to a problem may be represented as a set of parameters (for example, the parameters that optimise a neural network). These parameters (known as *genes*) are joined together to form a string of values (often referred to as a *chromosome*. For example, if our problem is to maximise a function of three variables, F(x; y; z), we might represent each variable by a 10-bit binary number (suitably scaled). Our chromosome would therefore contain three genes, and consist of 30 binary digits. The set of parameters represented by a particular chromosome is referred to as a *genotype*. The genotype contains the information required to construct an organism which is referred to as the *phenotype*. For example, in a bridge design task, the set of parameters specifying a particular design is the genotype, while the finished construction is the phenotype.

The fitness of an individual depends on the performance of the phenotype. This can be inferred from the genotype, i.e. it can be computed from the chromosome, using the fitness function. Assuming the interaction between parameters is nonlinear, the size of the search space is related to the number of bits used in the problem encoding. *For a bit string encoding of length L; the size of the search space is* 2^L *and forms a hypercube*. The genetic algorithm samples the corners of this L-dimensional hypercube. Generally, most test functions are at least 30 bits in length; anything much smaller represents a space which can be *enumerated.* Obviously, the expression 2^L grows exponentially. *As long as the number of "good solutions" to a problem are sparse with respect to the size of the search space, then random search or search by enumeration of a large search space is not a practical form of problem solving*. On the other hand, any search other than random search imposes some bias in terms of how it looks for better solutions and where it looks in the search space. A genetic algorithm belongs to the class of methods known as "*weak methods*" because it makes *relatively few assumptions about the problem that is being solve*d.

Genetic algorithms are often described as a global search method that does not use gradient information. Thus, nondifferentiable functions as well as functions with multiple local optima represent classes of problems to which genetic algorithms might be applied. Genetic algorithms, as a weak method, are robust but very general.

2.3 Fitness function

A fitness function must be devised for each problem to be solved. Given a particular chromosome, the fitness function returns a single numerical "fitness," or "figure of merit," which is supposed to be proportional to the "utility" or "ability" of the individual which that chromosome represents. For many problems, particularly function optimisation, the fitness function should simply measure the value of the function.

2.4 Reproduction

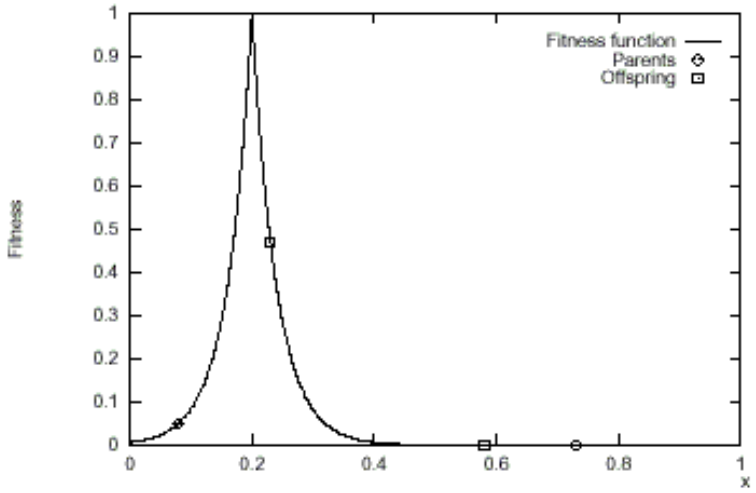
Good individuals will probably be selected several times in a generation, poor ones may not be at all. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. The previous c*rossover* example is known as *single point crossover*. Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, offspring are produced simply by duplicating the parents. This gives each individual a chance of passing on its genes without the disruption of crossover.

Mutation is applied to each child individually after crossover. It randomly alters each gene with a small probability. The next diagram shows the fifth gene of a chromosome being mutated:

The traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and helps ensure that no point in the search has a zero probability of being examined.

An example of two individuals reproducing to give two offspring is shown below:

Individual	Value	Fitness	Chromosome
Parent 1	0.08	0.05	00 01010010
Parent 2	0.73	0.000002	10 11101011
Offspring 1	0.23	0.47	00 11101011
Offspring 2	0.58	0.00007	10 01010010



The fitness function is an exponential function of one variable, with a maximum at x = 0:2. It is coded as a 10-bit binary number. This illustrates how it is possible for crossover to recombine parts of the chromosomes of two individuals and give rise to offspring of higher fitness. (Crossover can also produce offspring of low fitness, but these will not be likely to get selected for reproduction in the next generation.)

2.5 Convergence

The fitness of the best and the average individual in each generation increases towards a global optimum. Convergence is the progression towards increasing uniformity. *A gene is said to have converged when 95% of the population share the same value. The population is said to have converged when all of the genes have converged.*
As the population converges, the average fitness will approach that of the best individual.

A GA will always be subject to stochastic errors. One such problem is that of *genetic drift.* Even in the absence of any selection pressure (i.e. a constant fitness function), members of the population will still converge to some point in the solution space. This happens simply because of the accumulation of stochastic errors. If, by chance, a gene becomes predominant in the population, then it is just as likely to become more predominant in the next generation as it is to become less predominant. If an increase in predominance is sustained over several successive generations, and the population is finite, then a gene can spread to all members of the population. Once o gene has converged in this way, it is fixed; crossover cannot introduce new gene values. This produces a ratchet effect, so that as generations go by, each gene eventually becomes fixed. *The rate of genetic drift therefore provides a lower bound on the rate at which a GA can converge towards the correct solution*. That is, if the GA is to exploit gradient information in the fitness function, *the fitness function must provide a slope sufficiently large to counteract any genetic drift*. The rate of genetic drift can be reduced by increasing the mutation rate. *However, if the mutation rate is too high, the search becomes effectively random*, so once again gradient information in the fitness function is not exploited.

3 Comparisons

3.1 Strengths
The power of GAs comes from the fact that the technique is robust and can deal successfully with a wide range of difficult problems.

GAs are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding "acceptably good" solutions to problems "acceptably quickly". Where specialised techniques exist for solving particular problems, they are likely to outperform GAs in both speed and accuracy of the final result.

Even where existing techniques work well, improvements have been made by hybridising them with a GA.

The basic mechanism of a GA is so robust that, within fairly wide margins, parameter settings are not critical.

3.2 Weaknesses

A problem with GAs is that the genes from a few comparatively highly fit (but not optimal) individuals may rapidly come to dominate the population, causing it to converge on a local maximum. Once the population has converged, the ability of the GA to continue to search for better solutions is effectively eliminated: crossover of almost identical chromosomes produces little that is new. Only mutation remains to explore entirely new ground, and this simply performs a slow, random search.

3.3 Comparison with other methods

Any efficient optimisation algorithm must use two techniques to find a global maximum: *exploration* to investigate new and unknown areas in the search space, and *exploitation* to make use of knowledge found at points previously visited to help find better points. These two requirements are contradictory, and a good search algorithm must find a tradeoff between the two.

Neural nets

Both GAs and neural nets are adaptive, learn, can deal with highly nonlinear models and noisy data and are robust, "weak" random search methods. They do not need gradient information or smooth functions. In both cases their flexibility is also a drawback, since they have to be carefully structured and coded and are fairly application-specific.

For practical purposes they appear to work best in combination: neural nets can be used as the prime modelling tool, with GAs used to optimise the network parameters.

Random Search

The brute force approach for difficult functions is a random, or an enumerated search. Points in the search space are selected randomly, or in some systematic way, and their fitness evaluated. This is a very unintelligent strategy, and is rarely used by itself.

Gradient methods

A number of different methods for optimising well-behaved continuous functions have been developed which rely on using information about the gradient of the function to guide the direction of search. If the derivative of the function cannot be computed, because it is discontinuous, for example, these methods often fail. Such methods are generally referred to as *hillclimbing*. They can perform well on functions with only one peak (*unimodal* functions). But on functions with many peaks, (multimodal functions), they suffer from the problem that the first peak found will be climbed, and this may not be the highest peak. Having reached the top of a local maximum, no further progress can be made.

Iterated Search
Random search and gradient search may be combined to give an *iterated hillclimbing* search. Once one peak has been located, the hillclimb is started again, but with another, randomly chosen, starting point. This technique has the advantage of simplicity, and can perform well *if the function does not have too many local maxima*. However, since each random trial is carried out in isolation, *no overall picture of the "shape" of the domain is obtained*. As the random search progresses, it continues to allocate its trials evenly over the search space. This means that it will still evaluate just as many points in regions found to be of low fitness as in regions found to be of high fitness. A GA, by comparison, starts with an initial random population, *and allocates increasing trials to regions of the search space found to have high fitness*. This is a disadvantage if the maximum is in a small region, surrounded on all sides by regions of low fitness. This kind of function is difficult to optimise by any method, and here the simplicity of the iterated search usually wins.

Simulated annealing
This is essentially a modified version of hill climbing. Starting from a random point in the search space, a random move is made. If this move takes us to a higher point, it is accepted. If it takes us to a lower point, it is accepted only with probability p(t), where t is time. The function p(t) begins close to 1, but gradually reduces towards zero, the analogy being with the cooling of a solid. Initially therefore, any moves are accepted, but as the "temperature" reduces, the probability of accepting a negative move is lowered. *Negative moves are essential sometimes if local maxima are to be escaped*, but obviously too many negative moves will simply lead us away from the maximum. Like the random search, however, simulated annealing only deals with one candidate solution at a time, and so *does not build up an overall picture of the search space*. No information is saved from previous moves to guide the selection of new moves.

4 Suitability

Most traditional GA research has concentrated in the area of *numerical function optimisation*. GAs have been shown to be able to outperform conventional optimisation techniques on difficult, discontinuous, multimodal, noisy functions. *These characteristics are typical of market data, so this technique appears well suited for our objective of market modelling and asset allocation*.

For asset allocation, combinatorial optimisation requires solutions to problems involving arrangements of discrete objects. This is quite unlike function optimisation, and different coding, recombination, and fitness function techniques are required.

There are many applications of GAs to learning systems, the usual paradigm being that of a classifier system. The GA tries to evolve (i.e. learn) a set of "if : : : then" rules to deal with some particular situation. This has been applied to *economic modelling* and *market trading* [2], once again our area of interest.

5 Practical implementation

5.1 Fitness function
Along with the *coding scheme* used, *the fitness* function is the most crucial aspect of any GA. Ideally, the fitness function should be smooth and regular, so that chromosomes with reasonable fitness are to chromosomes with slightly better fitness. They should not have too many local maxima, or a very isolated global maximum. It should reflect the value of the chromosome in some "real" way, but unfortunately *the "real" value of a chromosome is not always a useful quantity for guiding genetic search*. In

combinatorial optimisation problems, where there are many constraints, most points in the search space often represent invalid chromosomes and hence have zero "real" value. Another approach which has been taken in this situation is to use a *penalty function*, which represents how poor the chromosome is, and *construct the fitness as (constant - penalty).* A suitable form is:

$f_a(x) = f(x) + M^k \mathbf{w}^T \mathbf{c}_v(\mathbf{x})$

where **w** is a vector of nonnegative weighting coefficients, the vector $\mathbf{c}_V$ quantifes the magnitudes of any constraint violations, M is the number of the current generation and k is a suitable exponent. The dependence of the penalty on generation number biases the search increasingly heavily towards feasible space as it progresses.

Penalty functions which represent the *amount* by which the constraints are violated are better than those which are based simply on the *number* of constraints which are violated. *Approximate function evaluation* is a technique which can sometimes be used if the fitness function is excessively slow or complex to evaluate. *A GA is robust enough to be able to converge in the face of the noise represented by the approximation. Approximate fitness techniques have to be used in cases where the fitness function is stochastic*.

5.2 Fitness Range Problems

Premature convergence

The initial population may be generated randomly, or using some heuristic method. At the start of a run, the values for each gene for different members of the population are randomly distributed. Consequently, there is a wide spread of individual fitnesses. As the run progresses, particular values for each gene begin to predominate. As the population converges, so the range of fitnesses in the population reduces. *This variation in fitness range throughout a run often leads to the problems of premature convergence and slow finishing.* Holland's *schema theorem* says that one should allocate reproductive opportunities to individuals in proportion to their relative fitness. But then premature convergence occurs because the population is not infinite. To make GAs work effectively on finite populations, the way individuals are selected for reproduction must be modified. One needs to control the number of reproductive opportunities each individual gets so that it is neither too large nor too small. The effect is to *compress the range of fitnesses*, and prevent any "super-fit" individuals from suddenly taking over.

Slow finishing

This is the *converse problem* to premature convergence. After many generations, the population will have largely converged, but may still not have precisely located the global maximum. The average fitness will be high, and there may be little difference between the best and the average individuals. Consequently *there is an insufficient gradient in the fitness function to push the GA towards the maximum.*
The same techniques used to combat premature convergence also combat slow finishing. They do this by *expanding the effective range of fitnesses* in the population. As with premature convergence, fitness scaling can be prone to overcompression due to just one "super poor" individual.

5.3 Parent selection techniques

Parent selection is the task of *allocating reproductive opportunities* to each individual. In principle, individuals from the population are copied to a "*mating pool*", with highly fit individuals being more likely to receive *more than one copy*, and unfit individuals being more likely to receive no copies. Under a strict generational replacement, the size of the mating pool is equal to the size of the population. After this, pairs of individuals are taken out of the mating pool at random, and mated. This is repeated until the

mating pool is exhausted. The behaviour of the GA very much depends on how individuals are chosen to go into the mating pool. Ways of doing this can be divided into two methods:

1) Explicit fitness remapping

To keep the mating pool the same size as the original population, the average of the number of reproductive trials allocated per individual must be one. If each individual's fitness is remapped by dividing it by the average fitness of the population, this effect is achieved. This remapping scheme allocates reproductive trials in proportion to raw fitness, according to Holland's theory. The remapped fitness of each individual will, in general, not be an integer. Since only an integral number of copies of each individual can be placed in the mating pool, we have to convert the number to an integer in a way that does not introduce bias. A widely used method is known as *stochastic remainder sampling without replacement*. A better method, *stochastic universal sampling* is elegantly simple and theoretically perfect. It is important not to confuse the sampling method with the parent selection method. Different parent selection methods may have advantages in different applications. *But a good sampling method is always good, for all selection methods, in all applications.*

Fitness scaling is a commonly employed method of remapping. The maximum number of reproductive trials allocated to an individual is set to a certain value, typically 2.0. This is achieved by subtracting a suitable value from the raw fitness score, then dividing by the average of the adjusted fitness values. Subtracting a fixed amount increases the ratio of maximum fitness to average fitness. Care must be taken to *prevent negative fitness values* being generated. However, the presence of *just one super-fit* individual (with a fitness ten times greater than any other, for example), can lead to *overcompression*. If the fitness scale is compressed so that the ratio of maximum to average is 2:1, then the rest of the population will have fitnesses clustered closely about 1. Although premature convergence has been prevented, it has been at the expense of effectively flattening out the fitness function. As mentioned above, if the fitness function is too flat, *genetic drift* will become a problem, so overcompression may lead not just to slower performance, but also to drift away from the maximum.

Fitness windowing is the same as fitness scaling, except the amount subtracted is the minimum fitness observed during the previous n generations, where n is typically 10. With this scheme the selection pressure (i.e. the ratio of maximum to average trials allocated) varies during a run, and also from problem to problem. The presence of a super-unfit individual will cause underexpansion, while super-fit individuals may still cause premature convergence, since they do not influence the degree of scaling applied. The problem with both fitness scaling and fitness windowing is that *the degree of compression is dictated by a single, extreme individual, either the fittest or the worst*. Performance will suffer if the extreme individual is *exceptionally* extreme.

Fitness ranking is another commonly employed method, which overcomes the reliance on an extreme individual. Individuals are sorted in order of raw fitness, and then reproductive fitness values are assigned according to rank. This may be done linearly or exponentially. This gives a similar result to fitness scaling, in that the ratio of the maximum to average fitness is normalised to a particular value. However, it also ensures that the remapped fitnesses of intermediate individuals are regularly spread out. Because of this, the effect of one or two extreme individuals will be negligible, irrespective of how much greater or less their fitness is than the rest of the population. The number of reproductive trials allocated to, say, the fifth best individual will always be the same, whatever the raw fitness values of those above (or below). The effect is that *overcompression ceases to be a problem.*
Several experiments have shown *ranking to be superior to fitness scaling*.

2. Implicit fitness remapping

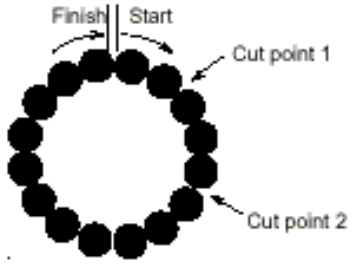
Implicit fitness remapping methods fill the mating pool without passing through the intermediate stage of remapping the fitness.

In *binary tournament selection*, pairs of individuals are picked at random from the population. Whichever has the higher fitness is copied into a mating pool (and then both are replaced in the original population). This is repeated until the mating pool is full. Larger tournaments may also be used, where the best of n randomly chosen individuals is copied into the mating pool. Using larger tournaments has the effect of increasing the selection pressure, since below-average individuals are less likely to win a tournament and vice-versa.

A further generalisation is *probabilistic binary tournament selection*. In this, the better individual wins the tournament with probability p, where $0.5 < p < 1$. Using lower values of p has the effect of decreasing the selection pressure, since below-average individuals are comparatively more likely to win a tournament and vice-versa. *By adjusting tournament size or win probability, the selection pressure can be made arbitrarily large or small.*

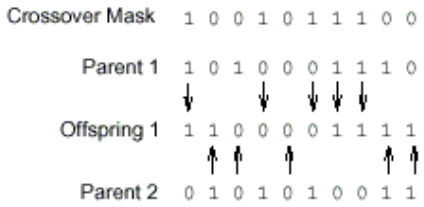
5.4 Other crossovers

2-point crossover
The problem with adding additional crossover points is that *building blocks are more likely to be disrupted*. However, an advantage of having more crossover points is that *the problem space may be searched more thoroughly.* In *2-point crossover*, (and multi-point crossover in general), rather than linear strings, chromosomes are regarded as *loops* formed by joining the ends together. To exchange a segment from one loop with that from another loop requires the selection of two cut points, as follows:



Here, 1-point crossover can be seen as 2-point crossover with one of the cut points fixed at the start of the string. Hence 2-point crossover performs the same task as 1-point crossover (i.e. exchanging a single segment), but is more general. A chromosome considered as a loop can contain more building blocks since they are able to "wrap around" at the end of the string. *2-point crossover is generally better than 1-point crossover.*

Uniform crossover
Uniform crossover is radically different to 1-point crossover. Each gene in the offspring is created by copying the corresponding gene from one or the other parent, chosen according to a randomly generated *crossover mask.*
Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask, the gene is copied from the second parent, as shown below.



The process is repeated with the parents exchanged to produce the second offspring. *A new crossover mask is randomly generated for each pair of parents.* Offspring therefore contain a mixture of genes from each parent. *The number of effective crossing points is not fixed, but will average L/2 (where L is the chromosome length).*

Uniform crossover appears to be more robust. Where two chromosomes are similar, the segments exchanged by 2-point crossover are likely to be identical, leading to offspring which are identical to their parents. This is less likely to happen with uniform crossover.

5.5 Inversion and Reordering

The order of genes on a chromosome is critical for the method to work effectively. Techniques for *reordering* the positions of genes in the chromosome during a run have been suggested. One such technique, *inversion*, works by reversing the order of genes between two randomly chosen positions within the chromosome. Reordering does nothing to lower epistasis (see below), but greatly expands the search space. Not only is the GA trying to find *good sets of gene values*, it is simultaneously trying to discover *good gene orderings* too.

5.6 Epistasis

Epistasis is the interaction between different genes in a chromosome. It is the extent to which the "*expression*" (i.e. contribution to fitness) of one gene depends on the values of other genes. The degree of interaction will be different for each gene in a chromosome. If a small change is made to one gene we expect a resultant change in chromosome fitness. This resultant change may vary according to the values of other genes.

5.7 Deception

One of the fundamental principles of GAs is that chromosomes which include schemata which are contained in the global optimum will increase in frequency (this is especially true of short, low-order schemata, known as building blocks). Eventually, via the process of crossover, these optimal schemata will come together, and the globally optimum chromosome will be constructed. *But if schemata which are not contained in the global optimum increase in frequency more rapidly than those which are, the GA will be misled, away from the global optimum, instead of towards it. This is known as deception*. Deception is a special case of epistasis and epistasis is necessary (but not sufficient) for deception. If epistasis is very high, the GA will not be effective. If it is very low, the GA will be outperformed by simpler techniques, such as hillclimbing.

5.8 Mutation and Naïve Evolution

Mutation is traditionally seen as a "background" operator, responsible for re-introducing *alleles* or inadvertently lost gene values, preventing genetic drift and providing a small element of random search in the vicinity of the population when it has largely converged. It is generally held that crossover is the main force leading to a thorough search of the problem space. *" Naïve evolution" (just selection and mutation) performs a hillclimb-like search which can be powerful without crossover.* However, mutation generally finds better solutions than a crossover-only regime. *Mutation becomes more productive, and crossover*

less productive, as the population converges. Despite its generally low probability of use, mutation is a very important operator. Its optimum probability is much more critical than that for

5.9 Niche and Speciation

Speciation is the process whereby a single species differentiates into two (or more) different species occupying different niches. In a GA, niches are analogous to maxima in the fitness function. Sometimes we have a fitness function which is known to be multimodal, and we may want to locate all the peaks. Unfortunately a traditional GA will not do this; the whole population will eventually converge on a single peak. This is due to *genetic drift*. The two basic techniques to solve this problem are to *maintain diversity*, or to *share the payoff* associated with a niche.

In *preselection*, offspring replace the parent only if the offspring's fitness exceeds that of the inferior parent. There is fierce competition between parents and children, so the payoff is not so much shared as fought over, and the winner takes all. This method helps to *maintain diversity (since strings tend to replace others which are similar to themselves)* and this helps prevent convergence on a single maximum.

In a *crowding* scheme, offspring are compared with a few (typically 2 or 3) randomly-chosen individuals from the population. The offspring replaces the most similar one found. This again aids diversity and indirectly encourages speciation.

5.10 Restricted Mating

The purpose of *restricted mating* is to encourage speciation, and reduce the production of *lethals*. A lethal is a child of parents from two different niches. Although each parent may be highly fit, the combination of their chromosomes may be highly unfit if it falls in the valley between the two maxima The general philosophy of restricted mating makes the assumption that if two similar parents (i.e. from the same niche) are mated, then the offspring will be similar. However, this will very much depend on the coding scheme and low epistasis. Under conventional crossover and mutation operators, two parents with similar genotypes will always produce offspring with similar genotypes. However, in a highly epistatic chromosome, there is no guarantee that these offspring will not be of low fitness, i.e. lethals.

The total reward available in any niche is fixed, and is distributed using a bucket-brigade mechanism. In *sharing*, several individuals which occupy the same niche are made to share the fitness payoff among them. Once a niche has reached its "carrying capacity", it no longer appears rewarding in comparison with other, unfilled niches.

5.11 Diploidy and Dominance

In the higher lifeforms, chromosomes contain two sets of genes, rather than just one. This is *diploidy*. (A *haploid* chromosome contains only one set of genes.) Virtually all work on GAs concentrates on haploid chromosomes. This is primarily for simplicity, although use of diploid chromosomes might have benefits. Diploid chromosomes lend advantages to individuals where the environment may change over a period of time. Having two genes *allows two different "solutions" to be remembered*, and passed on to offspring. One of these will be *dominant* (that is, it will be expressed in the phenotype), while the other will be *recessive*. If environmental conditions change, the dominance can shift, so that the other gene is dominant. This shift can take place *much more quickly than would be possible if evolutionary mechanisms had to alter the gene*. This mechanism is ideal if the environment *regularly switches between two states.*

6 Summary

The major advantage of genetic algorithms is their flexibility and robustness as a global search method. They are "weak methods" which do not use gradient information and make relatively few assumptions about the problem being solved. They can deal with highly nonlinear problems and non-differentiable functions as well as functions with multiple local optima. They are also readily amenable to parallel implementation, which renders them usable in real-time.

The primary drawback of genetic algorithms results from their flexibility. The designer has to come up with encoding schemes that allow the GA to take advantage of the underlying building blocks. One has to make sure the evaluation function assigns meaningful fitness measures to the GA. It is not always clear how the evaluation function can be formulated for the GA to produce an optimal solution. GAs are also computationally intensive and convergence is sometimes a problem.

As with neural nets, GAs look highly promising for modelling financial time series and asset allocation problems, because the driving variables are highly nonlinear, noisy, chaotic and changing all the time. (They are, however, well-established and relatively few.) They are also applicable in the refining and optimisation of technical rule-based market trading rules. Genetic algorithms may also be extremely useful if applied in conjunction with neural networks.

7 References

1. Holland, J.H., "Adaptation in Natural and Artificial Systems", MIT Press, 1975.
2. Deboeck, G. J. [Ed.]," Trading On The Edge", Wiley, 1994.
3. Whitely, D., "A Genetic Algorithm Tutorial", Technical Report CS-93-103, Colorado State University, 1993.