An Overview of Genetic Algorithms : Part 1, Fundamentals

David Beasley*
Department of Computing Mathematics,
University of Cardiff, Cardiff, CF2 4YN, UK

David R. Bull†
Department of Electrical and Electronic Engineering,
University of Bristol, Bristol, BS8 1TR, UK

Ralph R. Martin‡
Department of Computing Mathematics,
University of Cardiff, Cardiff, CF2 4YN, UK

University Computing, 1993, **15**(2) 58–69.

1 Introduction

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", first clearly stated by Charles Darwin in *The Origin of Species*. By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, if they have been suitably encoded. For example, GAs can be used to design bridge structures, for maximum strength/weight ratio, or to determine the least wasteful layout for cutting shapes from cloth. They can also be used for online process control, such as in a chemical plant, or load balancing on a multi-processor computer system.

The basic principles of GAs were first laid down rigourously by Holland [Hol75], and are well described in many texts (e.g. [Dav87, Dav91, Gre86, Gre90, Gol89a, Mic92]). GAs simulate those processes in natural populations which are essential to evolution. Exactly which biological processes are *essential* for evolution, and which processes have little or no role to play is still a matter for research; but the foundations are clear.

In nature, individuals in a population compete with each other for resources such as food, water and shelter. Also, members of the same species often compete to attract a mate. Those individuals which are most successful in surviving and attracting mates will have relatively larger numbers of offspring. Poorly performing individuals will produce few of even no offspring at all. This means that the genes from the highly adapted, or "fit" individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from different ancestors can sometimes produce "superfit" offspring, whose fitness is greater than that of either parent. In this way, species evolve to become more and more well suited to their environment.

GAs use a direct analogy of natural behaviour. They 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. For example, the fitness score might be the strength/weight ratio for a given bridge design. (In nature this is equivalent to assessing how effective an organism is at competing for resources.) The highly fit individuals are given opportunities to "reproduce", by "cross breeding" with other

*email: David.Beasley@cm.cf.ac.uk
†email: Dave.Bull@bristol.ac.uk
‡email: Ralph.Martin@cm.cf.ac.uk

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, being mixed and exchanged with other good characteristics as they go. 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.

GAs are not the only algorithms based on an analogy with nature. *Neural networks* are based on the behaviour of neurons in the brain. They can be used for a variety of classification tasks, such as pattern recognition, machine learning, image processing and expert systems. Their area of application partly overlaps that of GAs. The use of GAs for the design of neural networks is a current research area [HS91]. *Simulated annealing* is a search technique which is based on physical, rather than biological processes, and this is described in Section 3.4.

The power of GAs comes from the fact that the technique is robust, and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. 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 out-perform GAs in both speed and accuracy of the final result. The main ground for GAs, then, is in difficult areas where no such techniques exist. Even where existing techniques work well, improvements have been made by hybridising them with a GA.

In Section 2 we outline the basic principles of GAs, then in Section 3 we compare GAs with other search techniques. Sections 4 and 5 describe some of the theoretical and practical aspects of GAs, while Section 6 lists some of the applications GAs have been applied to.

Part 2 of this article will appear in the next issue of this journal. This will go into more detail, and discuss the problems which GA designers must address when faced with very difficult problems. We will also show how the basic GA can be improved by the use of problem-specific knowledge.

2 Basic Principles

The standard GA can be represented as shown in Figure 1.

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. These aspects are described below.

2.1 Coding

It is assumed that a potential solution to a problem may be represented as a set of parameters (for example, the dimensions of the beams in a bridge design). These parameters (known as *genes*) are joined together to form a string of values (often referred to as a *chromosome*). (Holland [Hol75] first showed, and many still believe, that the ideal is to use a binary alphabet for the string. Other possibilities will be discussed in Part 2 of this article.) 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.

In genetics terms, 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*. The same terms are used in GAs. 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.

```
BEGIN /* genetic algorithm */
    generate initial population
    compute fitness of each individual

    WHILE NOT finished DO
    BEGIN /* produce new generation */

        FOR population_size / 2 DO
        BEGIN /* reproductive cycle */
            select two individuals from old generation for mating
                /* biassed in favour of the fitter ones */
            recombine the two individuals to give two offspring
            compute fitness of the two offspring
            insert offspring in new generation
        END

        IF population has converged THEN
            finished := TRUE

    END

END
```

Figure 1: A Traditional Genetic Algorithm

2.2 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, it is obvious what the fitness function should measure—it should just be the value of the function. But this is not always the case, for example with combinatorial optimisation. In a realistic bridge design task, there are many performance measures we may want to optimise: strength/weight ratio, span, width, maximum load, cost, construction time—or, more likely, some combination of all these.

2.3 Reproduction

During the reproductive phase of the GA, individuals are selected from the population and recombined, producing offspring which will comprise the next generation. Parents are selected randomly from the population using a scheme which favours the more fit individuals. 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 most basic forms of these operators are as follows:

Crossover takes two individuals, and cuts their chromosome strings at some randomly chosen position, to produce two "head" segments, and two "tail" segments. The tail segments are then swapped over to produce two new full length chromosomes (see Figure 2). The two offspring each inherit some genes from each parent. This 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 (typically 0.001). Figure 3 shows the fifth gene of the 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

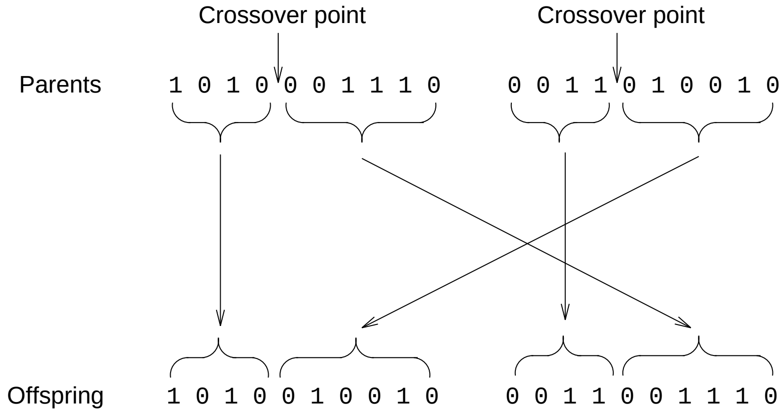

Figure 2: Single-point Crossover

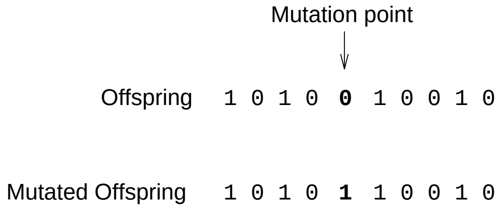

Figure 3: A single mutation

space has a zero probability of being examined. (An alternative point of view is explored in Part 2 of this article.)

An example of two individuals reproducing to give two offspring is shown in Figure 4. 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. Table 1 shows two parents and the offspring they produce when crossed over after the second bit (for clarity, no mutation is applied). 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. (Of course, crossover can also produce offspring of low fitness, but these will not be likely to get selected for reproduction in the next generation.)

2.4 Convergence

If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the 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 [DeJ75]. The *population* is said to have converged when all of the genes have converged.

Figure 5 shows how fitness varies in a typical GA. As the population converges, the average fitness will approach that of the best individual.

3 Comparison with other techniques

A number of other general purpose techniques have been proposed for use in connection with search and optimisation problems. Like a GA, they all assume that the problem is defined by a fitness function, which must be maximised. (All techniques can also deal with minimisation tasks—but to avoid confusion we will assume, without loss of generality, that maximisation is the aim.)

There are a great many optimisation techniques, some of which are only applicable to limited domains, for example, dynamic programming [Bel57]. This is a method for solving multi-step control problems which is only applicable where the overall fitness function is the sum of the fitness functions for each stage of the problem, and there is no interaction between stages. Some of the more general techniques are described below.

Individual	x	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

Table 1: Details of individuals in Figure 4

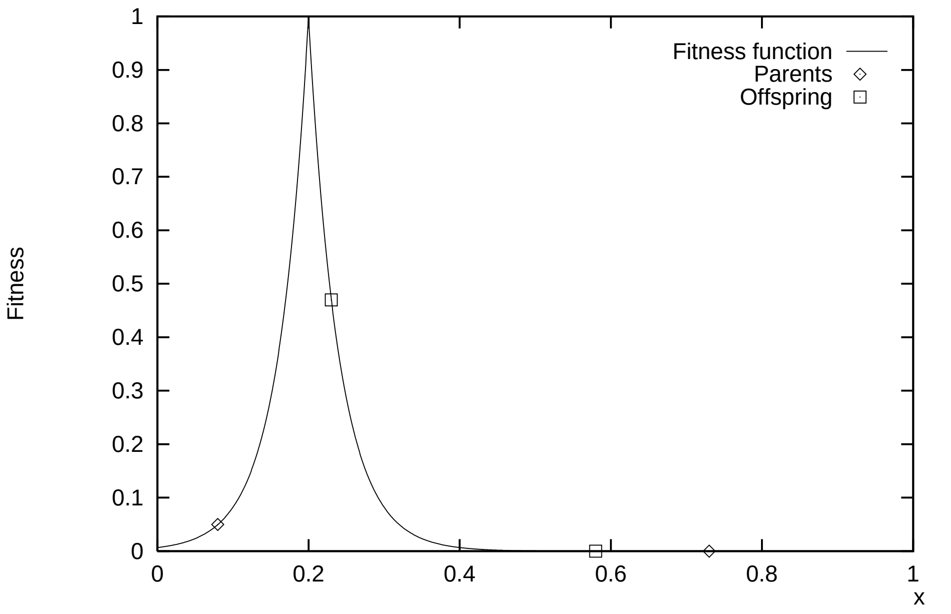

Figure 4: Illustration of crossover

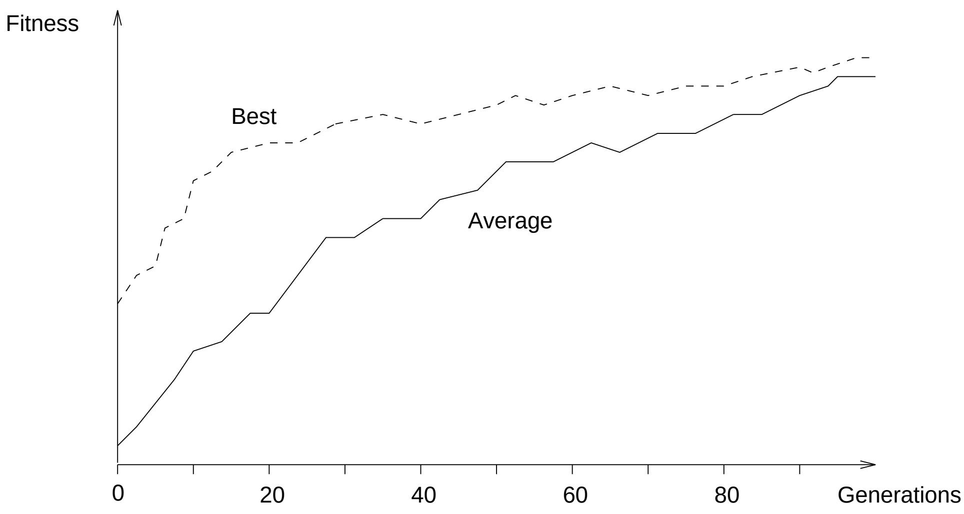

Figure 5: A Typical GA Run

3.1 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.

3.2 Gradient methods

A number of different methods for optimising well-behaved continuous functions have been developed [Bun84] 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. A 1-dimensional example is shown in Figure 6. The hillclimb starts from a randomly-chosen starting point, X. "Uphill" moves are made, and the peak at B is located. Higher peaks at A and C are not found.

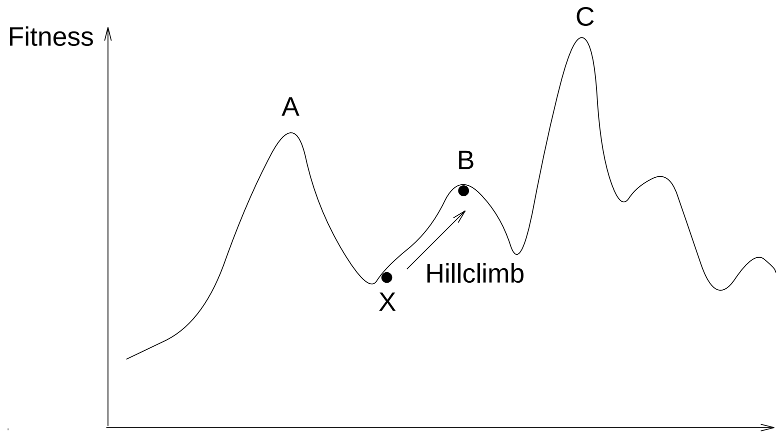

Figure 6: The hillclimbing problem

3.3 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 the day [Ack87].

3.4 Simulated annealing

This technique was invented by Kirkpatrick in 1982, and a good overview is given in [Rut89]. It 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. This technique is still the topic of much active research (e.g. fast re-annealing, parallel annealing), and it has been used successfully in many applications, for example, VLSI circuit layout [Rut89].

4 Why GAs work

Most research into GAs has so far concentrated on finding empirical rules for getting them to perform well. There is no accepted "general theory" which explains exactly why GAs have the properties they do. Nevertheless, several hypotheses have been put forward which can partially explain the success of GAs. These can be used to help us implement good GA applications.

4.1 Schemata and the Schema theorem

Holland's *schema theorem* [Hol75] was the first rigourous explanation of how GAs work. A schema is a pattern of gene values which may be represented (in a binary coding) by a string of characters in the alphabet {0 1 #}. A particular chromosome is said to contain a particular schema if it matches that schemata, with the "#" symbol matching anything. So, for example, the chromosome "1010" contains, among others, the schemata "10##," "#0#0," "##1#" and "101#." The *order* of a schema is the number of non-# symbols it contains (2, 2, 1, 3 respectively in the example). The *defining length* of a schema is the distance between the outermost non-# symbols (2, 3, 1, 3 respectively in the example).

The **schema theorem** explains the power of the GA in terms of how schemata are processed. Individuals in the population are given opportunities to reproduce, often referred to as *reproductive trials*, and produce offspring. The number of such opportunities an individual receives is in proportion to its fitness—hence the better individuals contribute more of their genes to the next generation. It is assumed that an individual's high fitness is due to the fact that it contains good schemata. By passing on more of these good schemata to the next generation, we increase the likelihood of finding even better solutions.

Holland showed that the optimum way to explore the search space is to allocate reproductive trials to individuals in proportion to their fitness relative to the rest of the population. In this way, good schemata receive an exponentially increasing number of trials in successive generations. This is called the schema theorem. He also showed that, since each individual contains a great many different schemata, the number of schemata which are effectively being processed in each generation is of the order n^3, where n is the population size. This property is known as **implicit parallelism**, and is one of the explanations for the good performance of GAs.

4.2 Building Block Hypothesis

According to Goldberg [Gol89a, p41], the power of the GA lies in it being able to find good *building blocks*. These are schemata of short defining length consisting of bits which work well together, and tend to lead to improved performance when incorporated into an individual. A successful coding scheme is one which encourages the formation of building blocks by ensuring that:

1. related genes are close together on the chromosome, while

2. there is little interaction between genes.

Interaction (often referred to as *epistasis*) between genes means that the contribution of a gene to the fitness depends on the value of other genes in the chromosome. (For example, for echo-location, bats must be able to generate ultrasonic squeaks, *and* have a good hearing system for detecting the echoes. The possession of either characteristic by itself is of little use. Therefore, the genes for good hearing can only increase the "fitness" of a bat if it also has genes for squeak production.)

In fact there is *always* some interaction between genes in multimodal fitness functions. This is significant because multimodal functions are the only sort of any real interest in GA research, since unimodal functions can be solved more easily using simpler methods.

If these rules are observed, then a GA will be as effective as predicted by the schema theorem.

Unfortunately, conditions (1) and (2) are not always easy to meet. Genes may be related in ways which do not allow all closely related ones to be placed close together in a one-dimensional string (for example, if they are related hierarchically). In many cases, the exact nature of the relationship between the genes may not be known to the programmer, so even if there are only simple relationships, it may still be impossible to arrange the coding to reflect this.

Condition (2) is a precondition for (1). If the contribution to overall fitness of each gene were independent of all other genes, then it would be possible to solve the problem by hillclimbing on each gene in turn. Clearly this is not possible in general. If we can ensure that each gene only interacts with a small number of other genes *and* these can be placed together on the chromosome, then conditions (1) and (2) can be met. But if there is a lot of interaction between genes, then neither condition can be met.

Clearly, we should *try* to design coding schemes to conform with Goldberg's recommendations, since this will ensure that the GA will work as well as possible. Two interesting questions therefore arise from this:

1. Is it possible, in general, to find coding schemes which fit the recommendations of the building block hypothesis? (And if so, then *how* can they be found?)

2. If it is *not* possible to find such ideal coding schemes, can the GA be modified to improve its performance in these circumstances? (And if so, *how*?)

These questions are both important research topics.

4.3 Exploration and exploitation

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.

A purely random search is good at exploration, but does no exploitation, while a purely hillclimbing method is good at exploitation, but does little exploration. Combinations of these two strategies can be quite effective, but it is difficult to know where the best balance lies (i.e. how much exploitation do we perform before giving up and exploring further?)

Holland [Hol75] showed that a GA combines both exploration and exploitation *at the same time* in an optimal way (using a k-armed bandit analogy, also described in [Gol89a, p36]). However, although this may be *theoretically* true for a GA, there are inevitably problems in practice. These arise because Holland made certain simplifying assumptions, including:

1. that population size is infinite,

2. that the fitness function accurately reflects the utility of a solution, and

3. that the genes in a chromosome do not interact significantly.

Assumption (1) can never be satisfied in practice. Because of this the performance of a GA will always be subject to stochastic errors. One such problem, which is also found in nature, is that of **genetic drift** [Boo87, GS87].

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 a 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.

Assumptions (2) and (3) can be satisfied for well-behaved laboratory test functions, but are harder to satisfy for real-world problems. Problems with the fitness function have been discussed above. Problems with gene interaction, (epistasis), have already been mentioned, and will be described further in Part 2.

5 Practical aspects of GAs

When designing a GA application, we need to consider far more than just the theoretical aspects described in the previous section. Each application will need its own fitness function, as mentioned earlier, but there are also less problem-specific practicalities to deal with. Most of the steps in the traditional GA (Figure 1) can be implemented using a number of different algorithms. For example, the initial population may be generated randomly, or using some heuristic method [Gre87, SG90].

In this section we describe different techniques for selecting two individuals to be mated. To understand the motivation behind these techniques, we must first describe the problems which they are trying to overcome. These problems are related to the fitness function, so first we shall look at this more closely.

5.1 Fitness function

Along with the coding scheme used, the fitness function is the most crucial aspect of any GA. Much research has concentrated on optimising all the other parts of a GA, since improvements can be applied to a variety of problems. Frequently, however, it has been found that only small improvements in performance can be made. Grefenstette [Gre86] sought an ideal set of parameters (in terms of crossover and mutation probabilities, population size, etc.) for a GA, but concluded that the basic mechanism of a GA was so robust that, within fairly wide margins, parameter settings were not critical. What *is* critical in the performance of a GA, however, is the fitness function, and the coding scheme used.

Ideally we want the fitness function to be smooth and regular, so that chromosomes with reasonable fitness are close (in parameter space) to chromosomes with slightly better fitness. For many problems of interest, unfortunately, it is not possible to construct such ideal fitness functions (if it were, we could simply use hill-climbing algorithms). Nevertheless, if GAs (or *any* search technique) are to perform well, we must find ways of constructing fitness functions which do not have too many local maxima, or a very isolated global maximum.

The general rule in constructing a fitness function is that it should reflect the value of the chromosome in some "real" way. As stated above, for many problems, the construction of the fitness function may be an obvious task. For example, if the problem is to design a fire-hose nozzle with maximum through flow, the fitness function is simply the amount of fluid which flows through the nozzle in unit time. Computing this may not be trivial, but at least we know *what* needs to be computed, and the knowledge of *how* to compute it can be found in physics textbooks.

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.

An example of such a problem is the construction of school timetables. A number of classes must be given a number of lessons, with 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 be effective in this case, we must invent a fitness function where the fitness of an invalid chromosome is viewed in terms of how good it is at *leading us towards* valid chromosomes. This, of course, is a Catch-22 situation. We have to know where the valid chromosomes are to ensure that nearby points can also be given good fitness values, and far away points given poor fitness values. But, if we don't know where the valid chromosomes are, this can't be done.

Cramer [Cra85] suggested that if the natural goal of the problem is all-or-nothing, better results can be obtained if we invent meaningful sub-goals, and reward those. In the timetable problem, for example, we might give a reward for each of the classes which has its lessons allocated in a valid way.

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) [Gol89a, p84]. Richardson *et al* [RPLH89] give some guidelines for constructing penalty functions. They say that those 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. Good penalty functions, they say, can be constructed from the *expected completion cost*. That is, given an invalid chromosome, how much will it "cost" to turn it into a valid one? DeJong & Spears [DS89] describe a method suitable for optimising boolean logic expressions. There is much scope for work in this area.

Approximate function evaluation is a technique which can sometimes be used if the fitness function is excessively slow or complex to evaluate. If a much faster function can be devised which approximately gives the value of the "true" fitness function, the GA may find a better chromosome in a given amount of CPU time than when using the "true" fitness function. If, for example, the simplified function is ten times faster, ten times as many function evaluations can be performed in the same time. An approximate evaluation of ten points in the search space is generally better than an exact evaluation of just one. A GA is robust enough to be able to converge in the face of the noise represented by the approximation. This technique was used in a medical image registration system, described by Goldberg [Gol89a, p138]. In attempting to align two images, it was found that optimum results were obtained when only 1/1000th of the pixels were tested.

Approximate fitness techniques *have* to be used in cases where the fitness function is stochastic. For example, if the problem is to evolve a good set of rules for playing a game, the fitness may be assessed by using them to play against an opponent. But each game will be different, so it is only ever possible to determine an *approximation* of the fitness of the rule set [Chi89]. Goldberg [Gol89a, p206–8] describes other techniques for approximate function evaluation, for example using an incremental computation based on the parents' fitness.

5.2 Fitness Range Problems

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*.

5.2.1 Premature convergence

A classical 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 [Gol89b].

The schema theorem says that we should allocate reproductive trials (or opportunities) to individuals *in proportion to their relative fitness*. But when we do this, premature convergence occurs—because the population is not infinite. In order to make GAs work effectively on *finite* populations, we must modify the way we select individuals for reproduction.

Ways of doing this are described in Section 5.3. The basic idea is 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.

5.2.2 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 (or, rather, underexpansion) due to just one "super poor" individual. These techniques are described below.

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 scheme (see Section 5.4), 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 types of methods. Firstly, we can take the fitness score of each individual, map it onto a new scale, and use this remapped value as the number of copies to go into the mating pool (the number of *reproductive trials*). Another method has been devised which achieves a similar effect, but without going through the intermediate step of computing a modified fitness. We shall call these methods *explicit fitness remapping* and *implicit fitness remapping*.

5.3.1 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.

Before we discuss other remapping schemes, there is a practical matter to be cleared up. 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 great deal of work has gone into finding the best way of doing this [Gol89a, p121]. A widely used method is known as *stochastic remainder sampling without replacement*. A better method, *stochastic universal sampling* was devised by Baker [Bak87], and 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 (such as Baker's) is always good, for all selection methods, in all applications.

As mentioned in Section 5.2.1, we do not want to allocate trials to individuals in direct proportion to raw fitness. Many alternative methods for remapping raw fitness, so as to prevent premature convergence, have been suggested. Several are described in [Bak85]. The major ones are described below.

Fitness scaling is a commonly employed method. In this, 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.

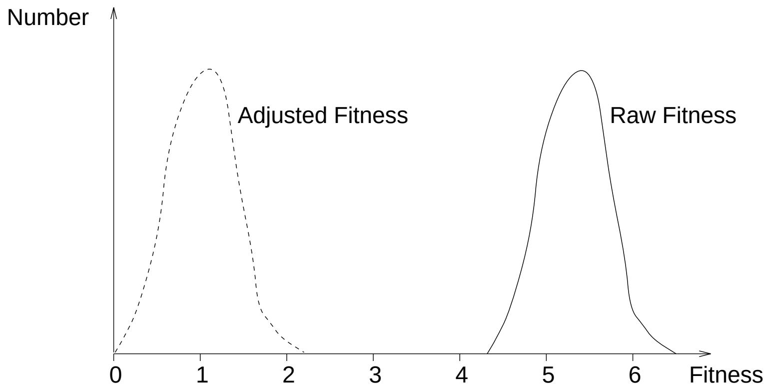

Figure 7: Raw and adjusted fitness histograms

Figure 7 shows a histogram of raw fitness values, with an average fitness of 5.4, and a maximum fitness of 6.5. This gives a maximum:average ratio of 1.2, so, without scaling, the most fit individual would be expected to receive 1.2 reproductive trials. To apply fitness scaling (perhaps *fitness shifting* would be a more accurate term) we subtract ($2 \times$ average $-$ maximum) $= 4.3$ from all fitnesses. This gives a histogram of adjusted fitnesses with an average of 1.1 and a maximum of 2.2, so the maximum:average ratio is now 2.

Fitness scaling tends to compress the range of fitnesses at the start of a run, thus slowing down convergence, and increasing the amount of exploration.

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 we have prevented premature convergence, we have done so 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 used in Grefenstette's *GENESIS* GA package [Gre84]. This is the same as fitness scaling, except the the amount to be subtracted is chosen differently. The *minimum* fitness in each generation is recorded, and 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 [Bak85], or exponentially [Dav89]. 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 [Bak85, Whi89].

Other methods (hybrid methods including using a dynamic population size) are described in [Bak85], but were found not to perform well.

5.3.2 Implicit fitness remapping

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

Tournament selection [Bri81, GD91] is such a technique. There are several variants. In the simplest, *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, while above average individuals are *more* likely to.

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, while above average individuals are *less* likely to.

By adjusting tournament size or win probability, the selection pressure can be made arbitrarily large or small.

Goldberg & Deb [GD91] compare four different schemes; proportionate selection, fitness ranking, tournament selection and steady state selection (see Section 5.4). They conclude that by suitable adjustment of parameters, all these schemes, (apart from proportionate selection), can be made to give similar performances, so there is no absolute "best" method.

5.4 Generation gaps and steady-state replacement

The generation gap is defined as the proportion of individuals in the population which are replaced in each generation. Most work has used a generation gap of 1—i.e. the whole population is replaced in each generation. This value is supported by the investigations of Grefenstette [Gre86]. However, a more recent trend has favoured steady-state replacement [Whi87, Whi89, Sys89, Dav89, Dav91]. This operates at the other extreme—in each generation only a few (typically two) individuals are replaced.

This may be a better model of what happens in nature. In short-lived species, including some insects, parents lay eggs, and then die before their offspring hatch. But in longer-lived species, including mammals, offspring and parents are alive concurrently. This allows parents to nurture and teach their offspring, but also gives rise to competition between them.

In the steady-state case, we not only have to consider how to select two individuals to be parents, but we also have to select two unlucky individuals from the population to be killed off, to make way for the offspring. Several schemes are possible, including:

1. selection of parents according to fitness, and selection of replacements at random
2. selection of parents at random, and selection of replacements by inverse fitness
3. selection of both parents and replacements according to fitness/inverse fitness

For example, Whitley's *GENITOR* algorithm [Whi89], selects parents according to their ranked fitness score, and the offspring replace the the two worst members of the population.

The essential difference between a conventional, generational replacement GA, and a steady state GA, is that population statistics (such as average fitness) are recomputed after *each* mating in a steady state GA, (this need not be computationally expensive if done incrementally), and the new offspring are immediately available for reproduction. Such a GA therefore has the opportunity to exploit a promising individual as soon as it is created.

However, Goldberg & Deb's investigations [GD91] found that the advantages claimed for steady-state selection seem to be related to the high initial growth rate. The same effects could be obtained, they claim, using exponential fitness ranking, or large-size tournament selection. They found no evidence that steady-state replacement is fundamentally better than generational.

6 Applications

Some example GA applications were mentioned in the introduction. To illustrate the flexibility of GAs, here we list some more. Some of these applications have been used in practice, while others remain as research topics.

Numerical function optimisation. Most traditional GA research has concentrated in this area. GAs have been shown to be able to outperform conventional optimisation techniques on difficult, discontinuous, multimodal, noisy functions [DeJ75].

Image processing. With medical X-rays or satellite images, there is often a need to align two images of the same area, taken at different times. By comparing a random sample of points on the two images, a GA can efficiently find a set of equations which transform one image to fit onto the other [Gol89a, p138].

A more unusual image processing task is that of producing pictures of criminal suspects [CJ91]. The GA replaces the role of the traditional photo-fit system, but uses a similar coding scheme. The GA generates a number of random faces, and the witness selects the two which are most similar to the suspect's face. These are then used to breed more faces for the next generation. The witness acts as the "fitness function" of the GA, and is able to control its convergence towards the correct image.

Combinatorial optimisation tasks require solutions to problems involving arrangements of discrete objects. This is quite unlike function optimisation, and different coding, recombination, and fitness function techniques are required. Probably the most widely studied combinatorial task is the **travelling salesperson** problem [Gol85, GS89, LHPM87]. Here the task is to find the shortest route for visiting a specified group of cities. Near optimal tours of several hundred cities can be determined. **Bin packing**, the task of determining how to fit a number of objects into a limited space, has many applications in industry, and has been widely studied [Dav85a, Jul92]. A particular example is the layout of VLSI integrated circuits [Fou85]. Closely related is **job shop scheduling**, or time-tabling, where the task is to allocate efficiently a set of resources (machines, people, rooms, facilities) to carry out a set of tasks, such as the manufacture of a number of batches of machine components [BUMK91, Dav85b, Sys91, WSF89]. There are obvious constraints: for example, the same machine cannot be used for doing two different things at the same time. The optimum allocation has the earliest overall completion time, or the minimum amount of "idle time" for each resource.

Design tasks can be a mix of combinatorial and function optimisation. We have already mentioned three design applications; bridge structure, a fire hose nozzle and neural network structure. GAs can often try things which a human designer would never have thought of—they are not afraid to experiment, and do not have preconceived ideas. Design GAs can be hybridised with more traditional optimisation or expert systems, to yield a range of designs which a human engineer can then assess.

Machine learning. 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 game playing [Axe87] and maze solving, as well as political and economic modelling [FMK91].

A major use of machine learning techniques has been in the field of **control** [DeJ80, Hun92, KG90]. In a large, complex system, such as a chemical plant, there may be many control parameters to be adjusted to keep the system running in an optimal way. Generally, the classifier system approach is used, so that rules are developed for controlling the system. The fitness of a set of rules may be assessed by judging their performance either on the real system itself, or on a computer model of it. Fogarty [Fog88] used the former method to develop rules for controlling the optimum gas/air mixture in furnaces. Goldberg modelled a gas pipeline system to determine a set of rules for controlling compressor stations and detecting leaks [Gol89a, p288]. Davis and Coombs used a similar approach to design communication network links [DC87].

7 Summary

GAs are a very broad and deep subject area, and most of our knowledge about them is empirical. This article has described the fundamental aspects of GAs; how they work, theoretical and practical aspects which underlie them, and how they compare with other techniques.

If this article has aroused your interest, you may wish to find out more. For those with access to the Usenet News system, the `comp.ai.genetic` newsgroup supports discussion about GA topics. A moderated bulletin, *GA-digest* is distributed by email from the US Navy's Artificial Intelligence Centre. Subscription is free. To join, send a request to: `GA-List-Request@aic.nrl.navy.mil` . They also support an FTP site, containing back issues of GA-digest, information on publications and conferences, and GA source code which can be freely copied. To use this service, connect using ftp to `ftp.aic.nrl.navy.mil` using *anonymous* as the user name and your email address as the password. Then change directory to `/pub/galist`. There is a `README` file which gives up-to-date information about the contents of the archive. The administrators request that you do *not* use this facility between 8am and 6pm EST (1pm to 11pm GMT), Monday to Friday.

Part 2 of this article will appear in a future issue of this journal, and will go into further detail.

References

[Ack87] D.H. Ackley. An empirical study of bit vector function optimization. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 13, pages 170–204. Pitman, 1987.

[Axe87] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 3, pages 32–41. Pitman, 1987.

[Bak85] J.E. Baker. Adaptive selection methods for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 101–111. Lawrence Erlbaum Associates, 1985.

[Bak87] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum Associates, 1987.

[Bel57] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[Boo87] L. Booker. Improving search in genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 5, pages 61–73. Pitman, 1987.

[Bri81] A. Brindle. *Genetic algorithms for function optimization.* PhD thesis, University of Alberta, 1981.

[BUMK91] S. Bagchi, S. Uckun, Y. Miyabe, and K. Kawamura. Exploring problem-specific recombination operators for job shop scheduling. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 10–17. Morgan Kaufmann, 1991.

[Bun84] B.D. Bunday. *Basic Optimisation methods.* Edward Arnold, 1984.

[Chi89] P-C. Chi. Genetic search with proportion estimates. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 92–97. Morgan Kaufmann, 1989.

[CJ91] C. Caldwell and V.S. Johnston. Tracking a criminal suspect through "face-space" with a genetic algorithm. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann, 1991.

[Cra85] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 183–187. Lawrence Erlbaum Associates, 1985.

[Dav85a] L. Davis. Applying adaptive algorithms to epistatic domains. In *9th Int. Joint Conf. on AI*, pages 162–164, 1985.

[Dav85b] L. Davis. Job shop scheduling with genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 136–140. Lawrence Erlbaum Associates, 1985.

[Dav87] L. Davis. *Genetic Algorithms and Simulated Annealing.* Pitman, 1987.

[Dav89] L. Davis. Adapting operator probabilities in genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.

[Dav91] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[DC87] L. Davis and S. Coombs. Genetic algorithms and communication link speed design: theoretical considerations. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 252–256. Lawrence Erlbaum Associates, 1987.

[DeJ75] K. DeJong. *The Analysis and behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.

[DeJ80] K. DeJong. Adaptive system design: a genetic approach. *IEE Trans SMC*, 10:566–574, 1980.

[DS89] K. DeJong and W.M. Spears. Using genetic algorithms to solve NP-complete problems. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 124–132. Morgan Kaufmann, 1989.

[FMK91] S. Forrest and G. Mayer-Kress. Genetic algorithms, nonlinear dynamical systems, and models of international security. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 13, pages 166–185. Van Nostrand Reinhold, 1991.

[Fog88] T.C. Fogarty. Rule-based optimization of combustion in multiple burner furnaces and boiler plants. *Engineering Applications of Artificial Intelligence*, 1(3):203–209, 1988.

[Fou85] M.P. Fourman. Compaction of symbolic layout using genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum Associates, 1985.

[GD91] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[Gol85] D.E. Goldberg. Alleles, loci, and the TSP. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, 1985.

[Gol89a] D.E. Goldberg. *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[Gol89b] D.E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79. Morgan Kaufmann, 1989.

[Gre84] J.J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.

[Gre86] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans SMC*, 16:122–128, 1986.

[Gre87] J.J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 4, pages 42–60. Pitman, 1987.

[Gre90] J.J. Grefenstette. Genetic algorithms and their applications. In A. Kent and J.G. Williams, editors, *Encyclopaedia of Computer Science and Technology*, pages 139–152. Marcel Dekker, 1990.

[GS87] D.E. Goldberg and P. Segrest. Finite markov chain analysis of genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 1–8. Lawrence Erlbaum Associates, 1987.

[GS89] M. Gorges-Schleuter. ASPARAGOS: an asychronous parallel genetic optimization strategy. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann, 1989.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[HS91] S.A. Harp and T. Samad. Genetic synthesis of neural network architecture. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 15, pages 202–221. Van Nostrand Reinhold, 1991.

[Hun92] K.J. Hunt. Polynimial LQG and h_∞ controller synthesis: a genetic algorithm aolution. In *Proc. IEEE Conf. Decision and Control*, pages –, 1992.

[Jul92] K. Juliff. Using a multi chromosome genetic algorithm to pack a truck. Technical Report RMIT CS TR 92-2, Royal Melbourne Institute of Technology, August 1992.

[KG90] K. Krishnakumar and D.E. Goldberg. Genetic algorithms in control system optimization. In *AIAA Guidance, Navigation, Control Conf.*, pages 1568–1577, 1990.

[LHPM87] G.E. Liepins, M.R. Hilliard, M. Palmer, and M. Morrow. Greedy genetics. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 90–99. Lawrence Erlbaum Associates, 1987.

[Mic92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

[RPLH89] J.T. Richardson, M.R. Palmer, G.E. Liepins, and M.R. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann, 1989.

[Rut89] R.A Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices Magazine*, pages 19–26, January 1989.

[SG90] A.C. Schultz and J.J. Grefenstette. Improving tactical plans with genetic algorithms. In *Proc. IEEE Conf. Tools for AI*, pages 328–344. IEEE Society Press, 1990.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1989.

[Sys91] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 21, pages 332–349. Van Nostrand Reinhold, 1991.

[Whi87] D. Whitley. Using reproductive evaluation to improve genetic search and heuristic discovery. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 108–115. Lawrence Erlbaum Associates, 1987.

[Whi89] D. Whitley. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[WSF89] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and travelling salesmen: The genetic edge recombination operator. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140. Morgan Kaufmann, 1989.