# Genetic Algorithms Coding Primer

A.I. Khan[†]

[†]*Lecturer in Computing and Information Technology*
*Heriot-Watt University, Riccarton,*
*Edinburgh, EH14 4AS, United Kingdom*

## Introduction

In this article the concept and background theory of genetic algorithm is introduced. The contents of this article are based on David Goldberg's book on Genetic Algorithms [1] and the MSc dissertation of Alan Dunsmore [2].

## 1 What is a Genetic Algorithm ?

It may be defined as a search algorithm based on the mechanics of natural selection and natural genetics. It is a combination of the process of *survival of the fittest* and structured yet randomized information exchange with some flair of intuitive search. An admissible design is represented as an individual and a set of such admissible designs termed as the population. The population of these designs or individuals is allowed to evolve over a number of generations under the control of the genetic algorithm. In every generation, a new set of individuals is created using bits and pieces of the fittest of the old generation; additionally an occasional new part is tried for good measure. While randomized, genetic algorithm is not a random walk. It efficiently exploits the historical information to speculate on new search points with expected improved performance.

The central theme of research on genetic algorithm has been *robustness*, the balance between efficiency and efficacy necessary for survival in many different environments. Designers of artificial systems - both software and hardware, whether engineering systems, computer systems or business systems can learn considerably from the efficiency, robustness, flexibility, fault tolerance and self-guidance of biological systems. Genetic algorithm attempts to model the natural process for the evolution of species thus including the desirable features such as robustness, fault tolerance an broad range efficiency in the search for problems defined in multi-dimensional space.

## 2 Contemporary Optimisation Methods and Genetic Algorithm

The contemporary optimisation methods may be classified broadly into three types; calculus based, enumerative and random based.

**Calculus based** methods have been looked into quite considerably for design optimisation. These methods suffer from two fundamental drawbacks. Firstly they are local in scope; the optima searched is confined to the best solutions available in the current neighbourhood of the starting point of the search in multi-dimensional space. Secondly the calculus based methods rely on the existence of the derivatives. These methods have an implicit requirement for smoothness and order in the problem defined parameter space. Unfortunately for the real world problems no such guarantees are available regarding the composure of the parameter space. The search spaces associated with real world problems may be fraught with discontinuities and comprise vast multi-modal, noisy search spaces.

**Enumerative schemes** are conceptually straightforward. The search space is descretised and defined. The search algorithm starts looking at the objective function values at every point discrete point in the space one at a time. Although the simplicity of this algorithm is attractive but these scheme suffer on account of being grossly inefficient. Most practical spaces are simply too large to be searched over all the points one at a time.

**Random schemes** or walks are based upon the selection of the search direction at random. The best results over a random walk are saved and then a new random search direction is chosen for searching purpose. These schemes do not come up with any guarantee of finding the best solution within reasonable time limits. Hence the question of efficient searching remains unanswered. However these scheme are capable of handling multi-modal noisy disjointed search spaces since they do not require gradient information such as the calculus based schemes.

## 2.1 Genetic Algorithm

Genetic algorithm is different from the above described optimisation schemes in four ways:

1. GA works with a coding of the parameter set, not the parameters themselves.
2. GA searches from a population of points, not a single point.
3. GA uses payoff (objective function) information, not derivatives or other auxiliary knowledge.
4. GA uses probabilistic transition rules, not deterministic rules.

Under many traditional optimisation methods, one would move from a single point in the parameter space to the next using some form of a deterministic transition rule to locate the next point. This strategy would however fail if the parameter space was multi-modal (many peaks) and the starting point was located within the premises of a shorter peak. The GA, in contrast, works from a rich database of points in the design space and has the ability to switch to a different peak subject to the payoff information and a certain factor of probability. Thus the GA may not be highly efficient in specific case, such as smooth uni-modal spaces, but provides an efficient and broad based searching ability over arbitrary shaped design search spaces. This aspect contributes to the robustness of the GA and resulting advantage over more commonly used techniques.

## 3 Similarity Templates (Schemata)

A very basic overview of the schemata processing and its effectiveness is provided here. If required a full and rigorous argument is given in reference [1].

In a search process given only payoff data (fitness value), we may look at the information that is contained in a population of strings and their objective function values which helps guide a directed search for improvement. Consider the string and the fitness values shown in Table 1.

| String | Fitness |
|---|---|
| 01101 | 169 |
| 11000 | 579 |
| 01000 | 64 |
| 10011 | 361 |

Table 1: A sample of Strings and Fitness

The string column in Table 1 represent the $x$ in its binary form and the Fitness column consists of $f(x)$ where $f(x) = x^2$. The objective is to maximize the function $f(x) = x^2$ over an interval of [0,31].

If we look at this table, scanning up and down the string column, certain similarities can be seen among the strings. We can see that certain string patterns seem highly associated with good performance. For example, in the sample population, the strings starting with a `1` seem to come up with best finesses. Two separate things may be noted from the above example. First we are seeking similarities among strings in the population. Second, so doing, we admit a wealth of new information to help guide a search. In order to see how much and precisely what information is admitted, the concept of schema (plural, schemata) or similarity templates must be considered.

A schema is a similarity template describing a subset of strings with similarities at certain string positions. Without loss of generality let us consider the binary alphabet 0,1. A special symbol, the * or wild card (don't care) symbol is required for full schema description. As an example, consider the strings and schemata of length 5. The schema *0000 matches two strings, namely $\{10000, 00000\}$. It should be emphasized that the * symbol is simply a notional device that allows description of all possible similarities among strings of a particular length and alphabet and is never explicitly processed by the genetic algorithm. In general, a particular string of length $l$ contains $2^l$ schemata since each position may take on its actual value or a don't care symbol. As a result, a population of size $n$ contains somewhere between $2^l$ and $n2^l$ schemata, depending upon the population diversity.

The counting argument shows that a wealth of information is contained in even moderately sized populations. But what does the GA do to this large number of diverse schemata; what is processed usefully and what is the resulting effect? These questions shall be dealt after the mechanism of the genetic algorithm is explained.

## 4 Implementation of a Simple Genetic Algorithm

Simplicity of operation and power of effect are two of the main attractions of the genetic algorithm approach. The mechanics of a simple genetic algorithm are surprisingly simple in concept and in computer implementation.

A simple genetic algorithm that yields good results in many practical problems is composed of three operators:

- Reproduction
- Crossover
- Mutation

### 4.1 Reproduction

Reproduction is a process in which individual strings are copied according to their objective function values, $f$. Intuitively, we may think of the function $f$ as some measure of gain or profit that we want to maximize. Copying strings according to their fitness values means that strings with higher values have a higher probability of contributing one or more offspring in the next generation. This operator is an artificial version of natural selection, a Darwinian survival of the fittest among strings creatures. Commonly the reproduction operator is implemented in algorithmic form by creating a biased roulette wheel. Suppose the sample population of strings created by a random coin toss has objective fitness function values $f$ as shown in Table 2. The process of random coin toss for the generation of a string is simply assigning [0,1] values at each binary position in the string after tossing an unbiased coin.

| No. | String | Fitness | % of Total |
|---|---|---|---|
| 1 | 01101 | 169 | 14.4 |
| 2 | 11000 | 576 | 49.2 |
| 3 | 01000 | 64 | 5.5 |
| 4 | 10011 | 361 | 30.9 |
| Total | | 1170 | 100.0 |

Table 2: Sample problem strings and fitness values

The fitness values in the Table 2 are calculated by decoding the binary values for the $x$ represented as strings to decimal and then evaluating the objective function ($f(x) = x^2$). These fitness values are used to calculate the percentage of population total fitness and then use these percentages to construct a weighted roulette wheel shown in Figure 1.

To reproduce the roulette wheel is spun, selecting a random number between 0 and 1, four times. Each time another offspring is required, a spin of the roulette wheel yields

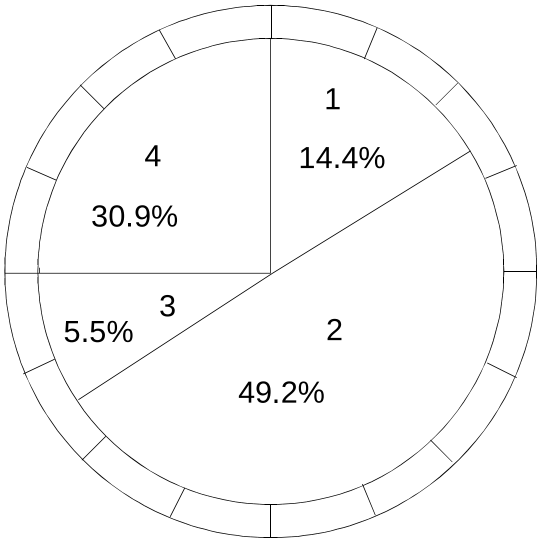

Figure 1: String selection using a biased roulette wheel [1]

the reproduction candidate. In this way, fitter strings have a higher number of offspring in the succeeding generation. An individual $i$ has a probability of selection given by $pselect_i = \frac{f_i}{\sum f}$. Once a string has been chosen for reproduction, an exact replica of the string is made. This string is then entered into a mating pool. From statistics, the roulette wheel selection should select a string $\frac{f_i}{\overline{f}}$ number of times in any generation, where $f_i$ being the fitness of the individual $i$ and $\overline{f}$ being the average fitness for the generation.

Other selection procedures exist which attempt to reduce the stochastic sampling error associated with the roulette wheel method. A popular method called *remainder stochastic selection without replacement* uses the $\frac{f_i}{\overline{f}}$ directly. Consider a value 1.56, the mating pool receives the number of copies of the design associated with the integer part of 1.56 i.e. one. Once the integer part of this value for all the designs have been used, designs are selected by looking at the designs within the generation and selecting designs using the fractional parts as the probability factors for the selection of the remaining designs for the mating pool. This ensures that better than average designs always receive at one copy for the mating pool.

## 5 Crossover

After the completion of entries for the mating pool the crossover stage proceeds. Members of the newly reproduced strings in the mating pool are selected and crossed over at random. The process of the crossover proceeds as follows: an integer position $k$ along the string is selected uniformly at random between 0 and the string length $l$. Two new strings are created by swapping all charters between positions 0 and $k$ inclusively. For example, consider strings $a_0$ and $a_1$ from our initial population:

$$\begin{array}{rcl} a_0 & = & 0 \,\big|\, 1 \;\; 1 \;\; 0 \;\; 1 \\ a_1 & = & 1 \,\big|\, 1 \;\; 0 \;\; 0 \;\; 0 \end{array}$$

Suppose in choosing a random number between 0 and 5, we obtain a $k = 4$ (as indicated by the separator symbol —). The resulting crossover yields two new strings $a_0'$ and $a_1'$ as part of a new generation:

$$\begin{array}{rcl} a_0' & = & 1 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \\ a_1' & = & 0 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \end{array}$$

The combined emphasis of reproduction and the structure, through random exchange of crossover, give genetic algorithm much of its power. Genetic algorithm efficiently exploits the wealth of information contained within the design strings by (1) reproducing high quality notions according to their performance and (2) crossing these notions with many other high-performance notions from from other strings. Thus, the action of crossover with previous reproduction speculates on the new ideas constructed from the high performance building blocks (notions) of past trials. Some researchers [3] have used multi-parameter crossover whereby each and every string undergoes crossover. Multi parameter crossover tend to change the fitness radically.

Every mating pair may not necessarily crossover. Crossover probability varies with the size of the population. For a population of 100, empirical observations show good results with a crossover probability of 0.6.

## 5.1 Mutation

In simple genetic algorithm, mutation is performed with a low probability factor for each character position on the design string. For binary coded string it implies changing the 1 to 0 and vice versa. Mutation is required to recover genetic material lost over the generation and may also be interpreted as a way to climb out of a local optima. Empirical observations show good results with probability of mutation kept at 0.003.

## 5.2 Objective function

Careful selection of the objective function plays an important feature in the efficiency of the genetic algorithm. Most real life design optimisation problems require imposition of design constraints. In GA based optimisation methods it is not possible to include the design constraints explicitly in the procedure. These design constraints are applied as *penalties* to the objective function.

The genetic algorithm is by nature a maximizing algorithm. It strives to increase the fitness of the individuals from generation to generation. Minimisation problems may however be formulated by maximizing an objective function, $f_{mp}$, give as,

$$f_{mp} \quad = \quad C - f_p$$

Where $C$ is taken as a large value which would keep the function value as positive. It however important to note that inordinately high value of $C$ may drastically reduce the efficiency of the genetic algorithm.

The fitness value $f_{mp}$ returned by the above fitness function is termed as the *raw fitness.*

## 5.3 Scaling of Raw Fitness

Regulation of the number of copies of different designs is especially important in small population GA search. At the start of a GA search it is common to have a few extraordinary individuals in a population of mediocre colleagues. If left to the roulette wheel selection rule ($pselect_i = \frac{f_i}{\sum f}$), the extraordinary individuals would take over a significant proportion of the population in a single generation. This is would in turn lead to premature convergence to a lot less than optimal. Later on during a run opposite is apparent. The average fitness of the population becomes very close to maximum fitness of the population. This leads to the best members getting the same number of copies for reproduction as the mediocre members. In both the cases the, at the beginning of the run and as the run matures, fitness scaling may help.

One useful scaling procedure is linear scaling. If we define $f$ as the raw fitness (given by $f_{mo}$) and the scaled fitness as $f'$ then linear scaling may be represented by the following relationship:

$$f' \quad = \quad af + b$$

The coefficient $a$ and $b$ may be chosen in a number of ways; however in all cases the average scaled fitness $f'_{avg}$ has to be equal to the average raw fitness $f_{avg}$. This is imperative since the subsequent use of the selection procedure will ensure that each average individual contributes one expected offspring to the next generation. To control the number of offspring given to the population member with maximum raw fitness, another scaling relationship is chosen to obtain a scaled maximum fitness, $'_{max} = C_{mult} f_{avg}$ where $C_{mult}$ is the number of expected copies for the best population members. For typical small populations ($n = 50$ to 100) a $C_{mult} = 1.2$ to 2.0 has been used successfully.

Towards the end of a run, this choice of $C_{mult}$ stretches the raw fitness significantly. Early on in a run, the normal scaling rule as shown in Figure 2 may be applied without any problem. The few extraordinary individuals get scaled down and the lowly members get scaled up. The more difficult situation is depicted in Figure 3. This situation occurs in a mature run. when a few bad design strings are far below the population average

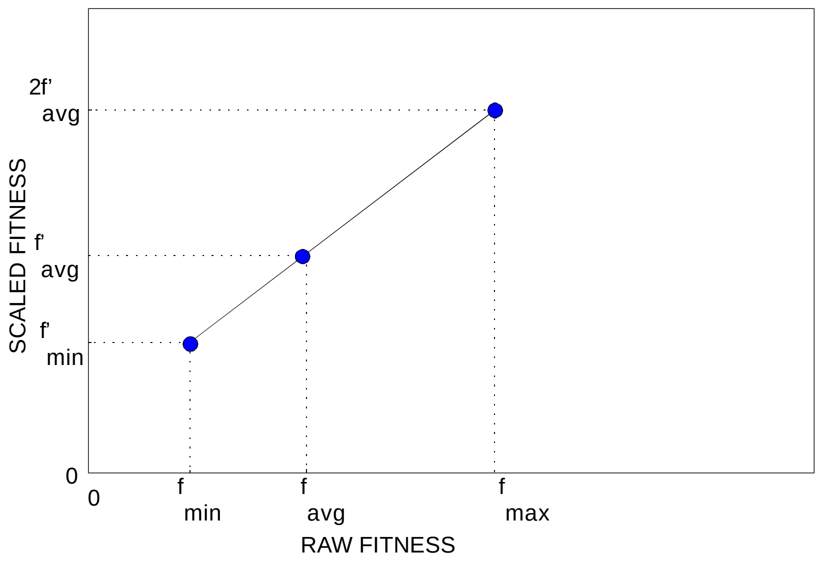

Figure 2: Linear scaling under normal conditions

and maximum and the average fitness of the population and the maximum fitness are relatively close together. If the scaling rule is applied in this situation, the stretching required on the relatively close average and maximum raw fitness values would cause the low fitness values to become negative after scaling. In this case it is not possible to scale to $C_{mult}$, the equality of the raw and the scaled average fitness is still maintained but the minimum raw fitness $f_{min}$ is scaled to $f'_{min} = 0$. One more complication may arise if a single individual manages to dominate a population completely; all members are the same. In this case the maximum, minimum and the average finesses would converge to the same value. Under this condition either the run could be terminated or scaling of the fitness may not be done.

## 6 Schemata Revisited

The questions raised at the end of the section 3 may now be answered since we have dealt with the general mechanism of genetic algorithm. It was asked that what does the GA do to the large number of diverse schemata; what is processed usefully and what is the resulting effect? Consider the effect of reproduction, crossover and mutation on the growth or decay of important schemata from generation to generation. The effect of reproduction on a particular schema is relatively easy to determine; since more highly fit strings have higher probabilities of selection, on average we give an increasing number of samples to the observed best similarity patterns. However, reproduction alone by itself doesn't sample new points in the search space. Crossover disrupts a schema with certain

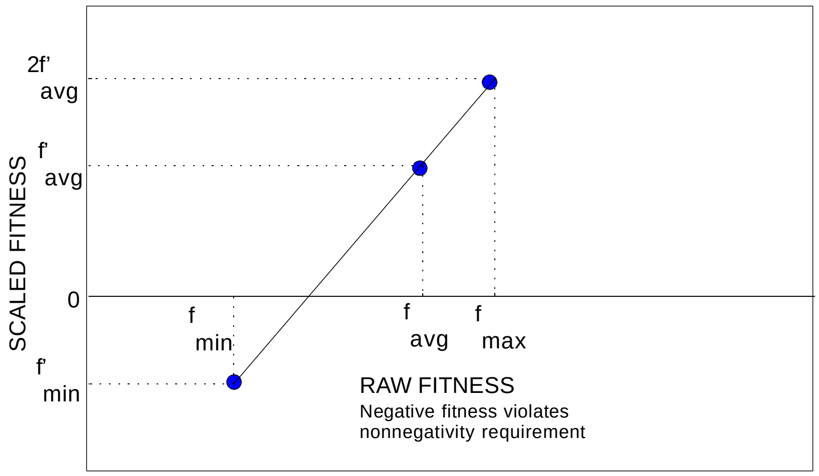

Figure 3: Difficulty with linear scaling procedure in a mature run. Points with low fitness can be scaled to negative values

degree of probability. For example consider two schemata 1***0 and **11*. The first is likely to be disrupted by a crossover, whereas the second relatively unlikely to be destroyed by the crossover. As a result the schemata of short defining length are left alone by the crossover and reproduce at a good sampling rate by the reproduction operator. Mutation at normal low rates does not disrupt a particular schema very frequently and this leads to an interesting conclusion. Highly fit, short-defining-length schemata are propagated generation to generation by exponentially increasing examples to the observed best.

To conclude it may be said that building blocks - short, high performance schemata are combined to form strings with expected higher performances. This occurs because building blocks are sampled at near optimal rates and recombined via crossover operator. Mutation has little effect on these building blocks; like an insurance policy, it helps prevent the irrevocable loss of potentially important genetic material. A GA discovers new solutions by speculating on many combinations of the best solutions within the current population.

## References

[1] Goldberg D. E., *"Genetic algorithms in search, optimization and machine learning"*, Addison-Wesley Publishing Company Inc 1989.

[2] Dunsmore A., *"The search for singularities in truss design using the genetic algorithms"*, MSc dissertation submitted to the Department of Civil and Offshore Engi-

neering of Heriot-Watt University, 1992.

[3] Jenkins W. M., *" Structural optimisation with the genetic algorithm"*, The Structural Engineer, vol 61, 418-422, 1991.