Introduction to GA's

Luke Abraham

I. THE START OF GA'S

This talk will give a brief introduction to the field of genetic algorithms (GA's), which were first suggested by John H Holland in his book *Adaption in Natural and Artificial Systems*, first published in 1975. He had noticed that simple representations (in his case, bit strings) could encode complicated structures, and that simple transformations could improve these structures.

GA's are often referred to as a *weak method*, but this also implies that it is not a powerful technique. The term is applied to methods which require few assumptions and can be used to solve a broad range of problems.

II. REPRESENTATION

Traditionally, the parameter values of GA's have been represented by bit strings, although this is no longer necessary. Arrays, trees and lists can be used, as well as strings with more than two members of the alphabet. However, there are a number of things to think about. I will only be talking about use of bit string implimentations in this talk.

Simpler, longer strings can encode more information that shorter, more complicated ones. Also, variables that are correlated should be as close as possible to each other on the string. If the "genes" are far apart on the string, any benefits derived from their correlation are likely to be lost in *crossover* (see section III C).

III. THE REQUIREMENTS OF A GA

There are four main steps in the updating of the population by generations. The size of the population is fixed, and each member of the population contains a complete set of parameters for the function being searched.

A. Fitness

This step evaluates the function being searched, and assigns a value to the individual member of the population upon which selection for reproduction is based. There are a large number of ways of scaling the fitness function to prevent early good (but not global) minimisations from dominating.

Some selection methods require that the fitness parameters be positive numbers, and the *scaling* would have to take this into account.

B. Selection for Reproduction

The members of the new population are selected based on their fitness. Again there are a large number of ways of selection, varying in complexity. Low selectivity accepts a large number of solutions, high selectivity will allow a few or one to dominate, however, a balance needs to be reached to prevent the solution from becoming trapped in a local minima.

One simple selection method is roulette selection: selection is proportional to fitness. The probability of an individual *i* being selected from its fitness f_i is

$$P_i = f_i / \sum_{j=1}^{N} f_j \tag{1}$$

A uniform random number would then be generated in the range 0 to 1. If this number is between the cumulative probabilities of the i^{th} and $(i+1)^{th}$ individuals, then the i^{th} individual is selected.

C. Crossover

This is where the real power of GA lies. The offspring of the two parents gets its parameter values from a random selection of of its parents values.

e.g., for two parents 11111111 and 0000000 the offspring could have 11100000 and 00011111 in a one point crossover at position 3 in the bit-string notation.

If the **11**** or **00**** were useful *schemata* (see section IV), then the crossover would have destroyed them. However, these would be more robust than ones such as *1****11.

D. Mutation

This would involve making a random change to one of the "genes" on the string, done randomly by a given probability.

e.g. 00011011 from the example above. This change may improve the individuals fitness, in which case it would be propagated to the next generation. If the mutation lowers the fitness it is unlikely to survive.

IV. SCHEMA

A string can be thought of as being made up of building blocks, or "sub-strings". In this *l*-bit string, a *schema* is defined as the subset of strings with similarities over a given number of positions, and a *similarity template* is a string taken over the underlying string alphabet together with a wild-card character (e.g. *) that matches any of the string characters.

So, for the similarity template 000** has schema 00000, 00001, 00010 & 00011. In general for binary strings of length *l* there are 2^l unique strings and 3^l schemata.

The number of fixed positions (the number of fixed positions) in any schema is known as its *order*, $o(H)$, so $o(*011*1*) = 4$, $o(*****1**) = 1$ etc.

A. Defining Length

If we have a string A=0111000, the schema H_1=*1****0 and H_2=***10** are represented within A. However, if a random crossover site were to occur at position 3, such that

A = 011|1000
H_1= *1*|***0
H_2= ***|10**

The defining length, $\delta(H)$ is defined as the position of the last fixed bit - the position of the first fixed bit, i.e. $\delta(H_1)$=5 and $\delta(H_2)$=1. We can use the defining length to determine the probability of any schema being destroyed in crossover

$$P_d = \delta(H)/\,(l-1) \tag{2}$$

So from the example above, $P_d(H_1) = 5/6$ and $P_d(H_2) = 1/6$. The survival rating will then be $P_s = 1 - P_d$, or, if the crossover itself is made by random choice, with probability P_c, then

$$P_s \geq 1 - P_c \cdot \delta(H)/(l-1) \tag{3}$$

B. Growth Equation

If we consider that at a given time-step, *t*, there are $m(H,t)$ examples of schema *H* in the population $\mathbf{A}(t)$. We known the probability of any string A_i being selected from (1), so we can calculate the number of representatives expected at time $(t+1)$

$$m(H,t+1) = m(H,t) \cdot N \cdot f(H)/\sum f_j \tag{4}$$

Where $f(H)$ is the average fitness of the strings representing H at time t. The average fitness of the entire population is $\overline{f} = \sum f_j/N$ then the above equation can be simplified

$$m(H,t+1) = m(H,t) \cdot f(H)/\overline{f} \tag{5}$$

i.e. a schema grows as the ratio of its average fitness to the average fitness of the population. If a schema has a fitness greater than the average populations it will have more samples in the next generation. If its fitness values are below the population average then it will receive a decreasing number of samples. Above-average schemata will grow and below-average will die off.

C. Mutation

Mutation is the random alteration of a single position with probability P_m, so the probability for a single bit to survive is $(1 - P_m)$. Since each mutation is statistically independent, a schema would survive when each of its $o(H)$ fixed positions survives, so the probability of a schema surviving mutation is $(1 - P_m)^{o(H)}$. If $P_m << 1$, the survival probability for mutation is

$$P_s = 1 - o(H) \cdot P_m \tag{6}$$

D. The Schema Theorem

By combining the three equations above ((3),(5),(6)) and ignoring small cross product terms then

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\overline{f}}[1 - P_c\frac{\delta(H)}{l-1} - o(H)P_m] \tag{7}$$

This is called the *Schema Theorem* or the Fundamental Theorem of Genetic Algorithms.

V. IMPLEMENTATION

Below (fig.1) is a simple flow diagram of the basic structure of a GA. The part that I have not mentioned before is the third step, because this is dependent on the problem being solved. In an abstract sense, this can be considered using the bit string notation (see table)

condition	string	matched
10**	1001	yes
10**	1011	yes
10**	1101	no

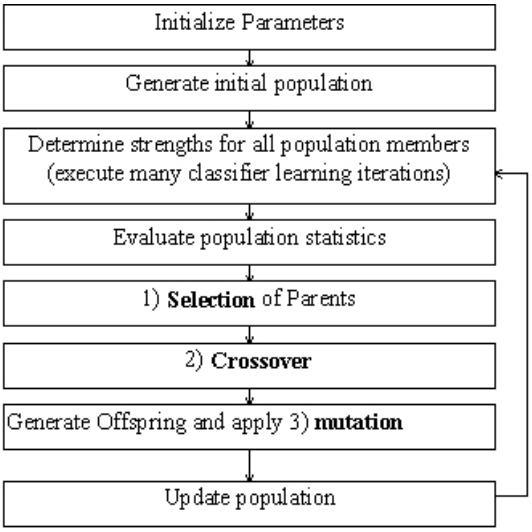


FIG. 1: A simple flow diagram of a GA

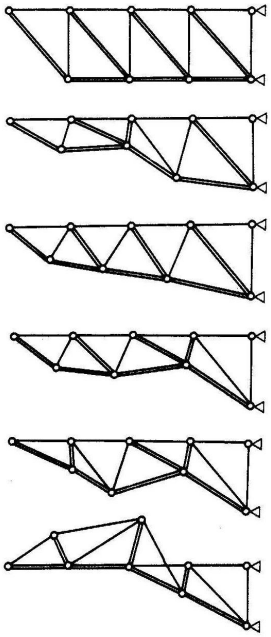
A. Clusters

When dealing with clusters, the determination of strengths of the population members is done from a calculation of each clusters potential energy. The lower the potential energy, the "stronger" the cluster. If f_i is the potential energy of cluster i if it is negative, and zero if it is positive. This is used to calculate the raw fitness from equation 1. This is then linearly scaled to $P_i' = aP_i + b$ so that $P_i' \geq 0$. The average of the raw fitness should be conserved.

Because GA's can be used for a wide range of different problems, and with a number of different representations, it might be best to look at a few examples.

B. Crane Jib design

This diagram (fig.2) is from the March 1995 edition of *Scientific Computing World*, showing the evolution of a crane jib design, where the only factor in fitness was the weight, so long as the jib would support a given load. The weight was reduced from 922 kg to 718 kg. However, the design itself isn't very practical.



Hoefler's evolution of a crane jib design. Members under compression are shown as thick lines, those under tension as thin ones. The projected design at the bottom is 20% lighter than the original at the top.

FIG. 2: Hoeflers Crane Jib

C. Optimisation of a function

This (fig. 3) is an example of the optimisation of a function with one local and one global minima (Neil Gersheneld, *The Nature of Mathematical Modelling* CUP). The height of the dots indicates the evolution of the population, with early generations lower (and more spread out), later generations higher up. As can be seen, these focus in on the global minima quite quickly.

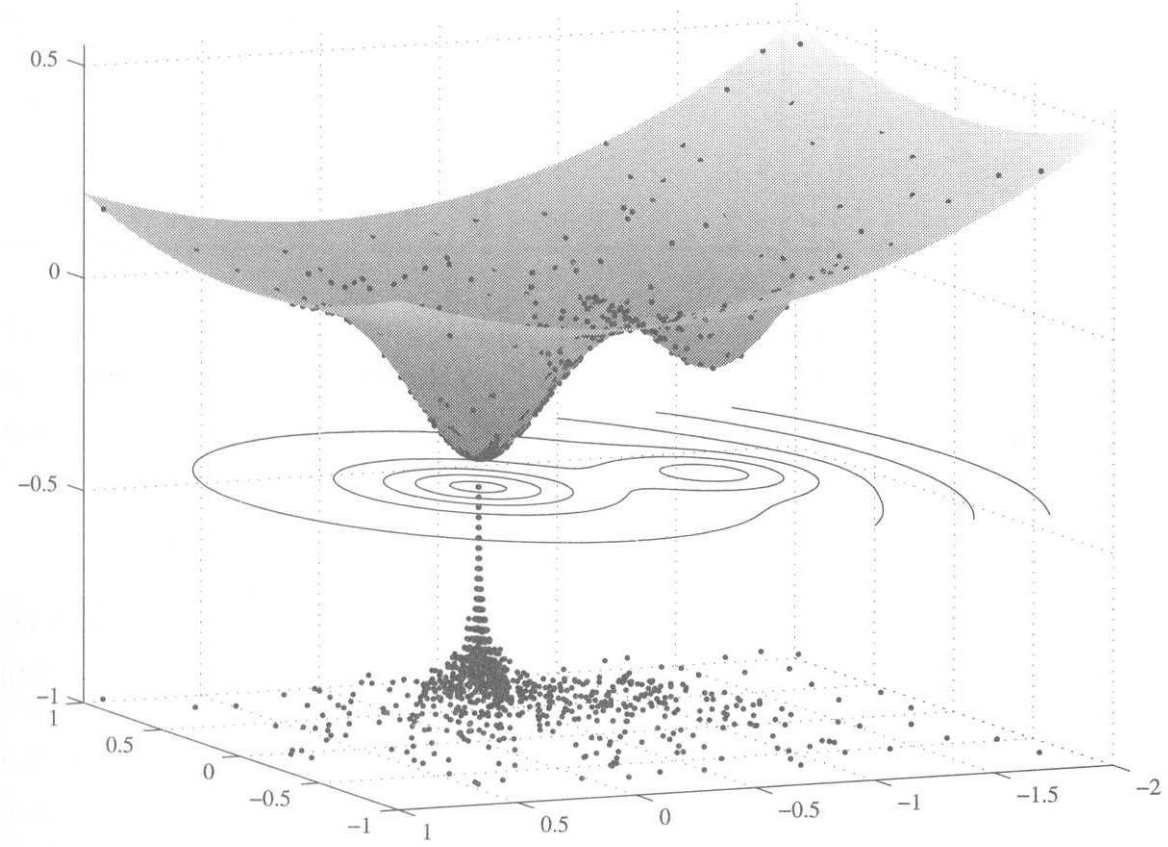


Figure 13.7. Optimization by a genetic algorithm, with populations displaced vertically at the bottom.

FIG. 3: Optimisation of a function

FIG. 4: hrrararharrarraharrararh