Genetic Algorithms for Optimizations

1. Introduction
Genetic Algorithms (GAs) are developed to mimic some of the processes observed in natural evolution. GAs use the concept of Darwin's theory of evolution to search for solutions for problems in a more "natural" way. Darwin also postulated that new breeds or classes of living things come into existence through the processes of reproduction, crossover, and mutation among existing organisms (Forrest, 1993). It is described the power of cumulative selection and its transformation of random process into non-random ones. In cumulative selection, each successive incremental improvement in a solution structure becomes the basis for the next generation. The principle of natural selection and population genetics is intrinsically powerful. Algorithms inspired by these principles have been successful when applied to optimization problems (Austin, 1990).

Regardless of their accuracy as biological development theories, the principle of natural selection and population genetics are powerful. Evolution strategies (or genetic based search algorithms) are very robust and will work in problem domains where traditional optimization algorithms will not. A genetic algorithm is a search procedure that uses random choice as a tool to guide a directed search in a search space.

2. Basic GA Cycle
GAs employ a random yet directed search for locating the globally optimal solution. Typically, GAs require a representation scheme to encode feasible solutions to the optimization problem. Usually this is done in the form of a string called a chromosome (or gene). Each gene represents one solution that is better or worse than other solutions in a population. An initial population of $n$ genes (for $n$ parameters) of length $L$ (the number of bits in each gene) is created. The genes are created in a random fashion, i.e., the values of the parameters that are coded in the genes are random values (created by randomly placing the 0s and 1s in the strings or any other coding system). This set of parameters is passed through a numerical model (fitness function) of the problem space. The numerical model gives out a solution based on the input set of parameters. On the basis of the quality of this solution, the gene is assigned a fitness
value. The fitness values are determined for each gene in the entire population of genes. Among all possible solutions, a fraction of the good solutions is selected, and the others are eliminated, to simulate the natural “survival of the fittest” process. The selected solutions undergo the processes of reproduction, crossover, and mutation to create a new generation of possible solutions (which are expected to perform better than the previous generation). The new set of genes is again decoded and evaluated, and a new generation is created using the three basic operators. This process is continued until convergence is achieved within a population (Ross, T.J. 1995).

Due to their perceived benefits, GAs have been successfully adopted to solve many science and engineering problems (Hegazy and Moselhi 1994). The benefit of this technique is that it searches for a solution from a broad spectrum of possible solutions, rather than restricts the search to a narrow domain where results would be normally expected. Genetic algorithms try to perform an intelligent search for a solution from a nearly infinite number of possible solutions.

A genetic algorithm consists of a string (genes) representation of nodes in the search space, a fitness function to evaluate the search nodes, and a stochastic assignment to control the genetic operations. The construction of a genetic algorithm for any problem can be separated into the following distinct and yet related tasks:

1. Determination of genes representation.
2. Determination of fitness function.
3. Determination of population size and number of generations.
These tasks will be briefly discussed below (Chan and Tansri, 1994).

3. Gene representation
To solve any optimization problem using a genetic algorithm, a coding scheme is necessary to encode the parameters of the problem into a gene. The scheme is
problem dependent and not unique. Most of the coding systems, a binary representation is used to encode the parameters.

4. Fitness Function
The fitness function is essentially the objective function for the problem. It provides a means of evaluating the search nodes (solutions). A well-defined fitness function is necessary to ensure success. The fitness value of each string is computed from the fitness function. A good string is one that scores a high fitness value. If the optimization problem is to minimize the function then maximize its negative or its inverse.

5. Population size and number of generations
One of the advantages of the genetic algorithm over traditional searching techniques is that it searches many nodes in the search space in parallel. The size of the parallel search is called the population size, which is equal to the number of strings (genes) in every generation. The population size is typically problem dependent and needs to be determined experimentally. Population size (number of genes) is also an important factor that affects the solution and the processing time it consumes. Larger population size (in the order of hundreds) increases the likelihood of obtaining a global optimum, however, substantially increases processing time.

The population is descended from one generation to the next in order to search for a better solution to a problem. The solutions will normally coverage to some (near) optimal points after a certain number of generations. The number of generations required to reach convergence is also a problem dependent. Therefore, it has to determined experimentally (Goldberg, 1989). Usually, the process is continued for a large number of offspring generations until an optimum gene is arrived at.

Once the gene structure and objective function (also referred to as fitness function) are set, the developed GA evolutionary procedure takes place on a population of parent genes. The simplest way to generate that population is randomly. Once the population is generated, the fitness of each gene in this population is evaluated and
accordingly its relative merit is calculated as the gene’s fitness divided by the total fitness of all genes.

A GA consists of the following steps:

1. Initialize a population of genes.
2. Evaluate each gene in the population.
3. Create new genes by mating current genes; apply reproduction, crossover and mutation.
4. Delete members of the population to make room for the new genes.
5. Evaluate the new genes and insert them into the population.
6. If the stopping criterion is satisfied, then stop and return the best gene; otherwise go to step 3.

Let’s suppose, there is a population with n individuals (genes):

Calculate the fitness value, evcr$(g_i)$ for each gene. $g_i = 1,2,3,...........,n$.

Find the total fitness of the population.

$$F = \sum_{i=1}^{n} evcr(g_i) \quad (3.1)$$

Calculate the probability (relative merit) of selection, $P_i$ for each gene $g_i$

$$P_i = \frac{evcr(g_i)}{F} \quad (3.2)$$

Calculate a cumulative probability $q_i$ for each gene $g_i = 1,2,3,...........,n$.

$$q_i = \sum_{j=1}^{i} p_j \quad (3.3)$$

6. Genetic Operators

6.1 Reproduction

Among the three genetic operators, reproduction is the process by which strings(genes) with better fitness values receive correspondingly better copies in the new generation, i.e. we try to ensure that better solutions persist and contribute to better offsprings (new strings) during successive generations. This is a way of ensuring the "survival of the
fittest'’ strings. Because the total number of strings in each generation is kept a constant (for computational economy and efficiency), strings with lower fitness values are eliminated. To do this process, use the roulette wheel n (population size) times. Each time generate a random number r between (0, 1). The \( i^{th} \) Gene is selected \( g_i \), where \( 2 \leq i \leq n \), such that \( q_{i-1} < r < q_i \). Using this process, the gene could be selected several times.

6.2 Crossover

The second operator, crossover, is the process in which the strings (genes) are able to mix and match their desirable qualities in a random fashion and it is the most important operator in genetic algorithms. Crossover (marriage) is by far a more common process and can be conducted by selecting two parent genes, exchanging their information, and producing an offspring. Each of the two parent genes is randomly selected in a manner such that its probability of being selected is proportional to its relative merit. This ensures that best genes have higher likelihood of being selected, without violating the diversity of the random process. Also, the exchange of information between the two parent genes is done through a random process (Austin, 1990).

![Figure 1: Crossover Operation to Generate Offspring](image)
The cross probability \( (P_c) \) is one of the major parameters in GA. \( P_c \) provides probability that \( P_c \times n \) structures undergo crossover operation. The higher the crossover rate, the more quickly new structures are introduced to the population. For each string, a random number \( r \) between \((0, 1)\) is generated. If \( r < P_c \), then this string is selected for crossover. The crossover proceeds in a simple way. For each couple of strings two random numbers are generated between \([1 \text{ and } m-1]\), where \( m \) is the length of the string and the portions of the strings between the two randomly selected locations in the two strings are exchanged as shown in Fig ( ), assuming that the crossover location occurs at positions 3 and 6 respectively. In this way information is exchanged and combined. Reproduction and crossover together give genetic algorithms most of their searching power.

6.3 Mutation

As opposed to crossover, which resembles the main natural method of reproduction (Goldberg 1989), mutation is a rare process that resembles the process of a sudden generation of an odd offspring that turns to be a genius. The benefit of the mutation process is that it can break any stagnation in the evolutionary process, avoiding local minimums. In order to understand the need for mutation, let us consider the case where reproduction or crossover may not be able to find an optimum solution to a problem. During the creation of a generation, it is possible that the entire population of strings is missing a vital bit information that is important for determining the correct or the most nearly optimum solution. Future generations that would be created using reproduction and crossover would be able to alleviate this problem. Here mutation becomes important. Occasionally, the value at a certain string location is changed. Mutation thus ensures that the vital bit of information is introduced into generation. Mutation, as it does in nature, takes place very rarely, on the order of once in a thousand bit string locations (a suggested mutation rate is 0.005/bit/generation (Forrest, 1993). The mutation probability \( (P_m) \) gives the expected number of mutated bits as:

\[
\text{Number of mutated bits} = P_m \times m \times n
\]

(3.4)

To do this, a random number " \( r \) " between \([0, 1]\) is generated, if \( r \) is smaller than \( P_m \) then mutate the bit, otherwise, go to another bit.