CHAPTER 4

GENETIC ALGORITHM

4.1 INTRODUCTION

Genetic Algorithms (GAs) were first proposed by John Holland (Holland 1975) whose ideas were applied and expanded on by Goldberg (Goldberg 1989). GAs is a heuristic search technique based on the principles of the Darwinian idea of survival of the fittest and natural genetics. Holland’s work was primarily an attempt to mathematically understand the adaptive processes of nature, but the general emphasis of GA research since then has been in finding applications, many in the field of combinatorial optimization. In this field, GAs must be seen as one of many possible heuristic techniques, many of which are described by Reeves (1993).

4.2 EVOLUTIONARY COMPUTATION

Natural evolution is a hypothetical population-based optimization process. Simulating this process on a computer results in stochastic optimization techniques that can often out-perform classical methods of optimization when applied to difficult real-world problems.

Figure 4.1 gives the description of the evolutionary processes used by the genetic algorithm. For iteration, the algorithm the processes of selection, reproduction and mutation each take place in order to produce the next generation of solutions.
4.3 DEFINITION OF GENETIC ALGORITHM

Genetic Algorithm (GA) is a search algorithm based on the conjecture of natural selection and genetics. The features of genetic algorithm are different from other search techniques in several aspects. First, the algorithm is a multi-path that searches many peaks in parallel, and hence reducing the possibility of local minimum trapping. Secondly, GA works with a coding of parameters instead of the parameters themselves. The coding of parameter will help the genetic operator to evolve the current state into the next state with minimum computations. Thirdly, GA evaluates the fitness of each string to guide its search instead of the optimization function. The genetic algorithm only needs to evaluate objective function (fitness) to guide its search. There is no requirement for derivatives or other auxiliary knowledge. Hence, there is no need for computation of derivatives or other auxiliary functions. Finally, GA explores the search space where the probability of finding improved performance is high (Lee and El-Sharkawi 2003).
4.4 THE ALGORITHM

GAs act on a population pool of proposed solutions, called chromosomes. The pool is generally maintained to be of constant size p. Each proposed solution is evaluated using a fitness function. The initial population pool is generally randomly selected. At each generation a possibly variable number of chromosome pairs are selected to act as parents. From each pair two offsprings are created using specified genetic operators. The new collection of offspring chromosomes are then merged back into the population pool in a way that maintains the constant size, p. This process continues until some specified finishing criterion is satisfied (Goldberg 1989). The decisions in implementing a GA for a particular problem are concerned with representation, fitness evaluation and the select, create and merge methods. The algorithm as described in (Smith 1995) is given in Figure 4.2. An o-schema is a set of permutations with a particular set of elements in common positions, for example, the permutation 1 4 2 3 5 is an instance of o-schema 1 * * * 5 where * can be any element not already specified. Two o-schemata are said to be compatible if it is possible for a chromosome to be an instance of both o-schemata. For example, 1 * * * 5 and 1 3 * * * are compatible, but 1 * * * 5 and 2 3 * * * are not. GAs act to combine short o-schemata with high average fitness to produce better solutions. Hence the type of problem on which a GA will be effective is one where “building blocks” can be combined to construct good solutions. Holland discusses in depth the reasons why GAs work.
### 4.4.1 Representation

The most obvious representation of a permutation is an array of \( q \) integers. The elements of a permutation will be indexed from 1 to \( q \), so that \( P[i] \) will denote the \( i^{th} \) element of a permutation, \( P \). The problem with this representation is that the traditional crossover techniques will not produce permutations. The data is represented as sequence of characters. For example: 

\[
\text{Parent1 : } [2 \ 1 \ 5 \ 4 \ 3 \ 6 \ | \ 7 \ 9 \ 8], \text{ parent2 : } [5 \ 3 \ 9 \ 6 \ 8 \ 2 \ | \ 1 \ 4 \ 7] \text{ after crossover, child1 : } [2 \ 1 \ 5 \ 4 \ 3 \ 6 \ | \ 1 \ 4 \ 7] \text{ is not permutation because 1, 4 elements are repeated.}
\]

In the section on `create_` various operators for the permutation representation are discussed, but other representations have also been proposed. Among these are the ordinal and adjacency representation.
(Grefenstette et al 1985) the positions listing representation (Poon and Carter 1995) and the precedence matrix representation (Fox and McMahon 1991). The ordinal and adjacency representations are generally accepted to be unsatisfactory for reasons given (Grefenstette et al 1985) and are not covered here. The position listing representation for any permutation, $P$ is simply the inverse, $P^{-1}$. The precedence matrix representation for a permutation is a $q \times q$ matrix, the manipulation of which is obviously more time consuming than operations involving a permutation. For each element of a permutation $P$, the precedence matrix, $M$, has an entry 1 in position $M[x, y]$ if and only if element $x$ occurs before element $y$ in permutation $P$. Otherwise the entry is 0.

4.4.2 Select Methods

The purpose of a select procedure is first to generate a discrete probability distribution function, Probability Density Function (PDF) over the population pool which is then randomly sampled for each parent required. The number of parents required at each generation is generally either fixed or an even random number between 2 and $p$. It is necessary that the better (higher if maximizing and lower if minimizing) the fitness of a chromosome the higher probability it should have of being chosen. Two methods of generating a PDF are described below.

• Roulette

Proposed by Goldberg (Goldberg 1989) roulette is the most commonly used selection procedure. Given a population of size $p$, the relative fitness of each chromosome is evaluated and used as the probability of selection. Let $X$ be a random variable defined on the population pool and
\( f \in \mathbb{R}^+ \) denote the fitness of the \( i^{th} \) member of the pool, \( x_i \). Then, for maximization problems,

\[
P(X = x_i) = \frac{f_i}{\sum_{i=1}^{p} f_i} \quad (4.1)
\]

\( P \) is the probability of selecting chromosome \( x_i \). For minimization problems the relative deviation from the maximum current fitness value is used as the fitness. Let \( f_{\text{max}} \) be the largest fitness value in the current population. Let \( f_i' = f_{\text{max}} - f_i \), then the probability of selecting chromosome \( x_i \) is given in equation (4.2).

\[
P(X = x_i) = \frac{f_i'}{\sum_{i=1}^{p} f_i'} \quad (4.2)
\]

Imagine a **roulette wheel** where all the chromosomes in the population are placed. Figure 4.3 shows that the size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome.

![Figure 4.3 The value of the fitness function for chromosomes](image)

A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value
will be selected more times. Variations of the basic roulette procedure have been suggested (Baker 1987).

- **Ranking**

  This method was introduced by Baker (Baker 1985). Selection by ranking, shifts the emphasis onto a prospective parent’s relative position in the population pool irrespective of actual fitness values. Let $p_i$ denote the rank of chromosome $x_i$ in the population pool, then the probability distribution used by Baker is defined as equation (4.3).

  \[
  P(X= x_i) = \frac{2p_i}{p(p+1)}
  \]  

  (4.3)

  An alternative distribution is given by equation (4.4)

  \[
  R_i = \frac{(p - p_i)^r}{\sum_{j=1}^{p} j^r}
  \]  

  (4.4)

  Where $r$ is a ranking parameter, which allows the weighting of importance of position. So $r = 0$ is equivalent to random selection, and setting $r > 1$ increases the probability of the top chromosomes being selected. Then the probability of selecting chromosome $x_i$ is given in equation (4.5)

  \[
  P(X= x_i) = \frac{R_i}{\sum_{j=1}^{p} R_j}
  \]  

  (4.5)
Another popular selection method, particularly in parallel GA implementations, is tournament selection (Muhlenbein 1989). Other selection methods include Fibonacci, exponential and tendency, all of which are simply different ways of ranking the population pool and exclusive, which only selects chromosomes from the top of the population pool. The previous type of selection will have problems when there are big differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses, then the other chromosomes will have very few chances to be selected (Marek Obitko 1998b). Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness $1$, the second worst $2$ etc. and the best will have fitness $N$ (number of chromosomes in population).

Figures 4.4 and 4.5 show the situation of fitness before and after ranking.
Now all the chromosomes have a chance to be selected. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

4.4.3 Create Method

The create stage of the algorithm requires defining a method of generating offspring of any two parents selected for mating which are also valid solutions. This is achieved using a crossover operator and mutation. Apart from the ordinal representation, all of the representations mentioned above require specialized crossover operators. The alternating *edges* and *subtour* chunking operators have been proposed for the adjacency representation (Grefenstette 1985), two tie breaking operators for the positions listing representation (Poon and Carter 1995) and the intersection and union crossover operators for the precedence matrix representation (Fox and McMahon 1991). In this section only operators for the permutation representation are described in detail. The operators described below are also valid for chromosomes defined by the positions listing representation, since these chromosomes are also permutations. Conversely, the tie breaking operators described by Poon and Carter (1995) are valid operators for the permutation representation. It is a problem specific issue as to which representation and operator to use. The tie breaking operators create a child by two point crossover and replace any duplicates with the missing elements randomly. Intersection crossover preserves the precedence relationships which are common to both parents and union crossover combines some precedence information from both parents. Poon and Carter give a method of performing union crossover without having to use the precedence matrix representation.
4.4.4 Crossover Operators

A good operator will combine the information from both parents with as little distortion as possible. If the permutation representation is used, the traditional methods of crossover used for combinatorial problems such as one point, two point and uniform cannot be applied, since the offspring will generally not be permutations. For this reason, several alternative operators have been suggested. Radcliffe (1991) describes the desirable properties of genetic operators in the general case. These properties, in relation to permutation problems, are reproduced below, but in assessing the usefulness of any permutation operator it is important to consider the nature of the problem. In some permutation problems the absolute position of the elements is the overriding concern. For example, in job-shop scheduling (Fang et al 1993) the elements represent jobs and their position on the chromosome represents the machine on which to perform the job and the order in which the job should be done. In other permutation problems, position relative to the other elements may be of more importance. It is argued that this is true of Travelling Salesman Problem (TSP) (Whitley et al 1989). Radcliffes informal analogy is reproduced here to help explain the properties.

Suppose the chromosomes are people and characteristics used to define a set of o-schemata are hair color and eye color.

- Respect. An operator is respectful to both parents if for parents \( P_1, P_2 \) and offspring \( C_1, C_2 \)
\[
P_1[i] = P_2[i] \Rightarrow C_1[i] = C_2[i] = P_1[i] \quad 0 < i \leq q.
\]

If both parents have blue eyes then their offspring must have blue eyes.
• Strictly Transmit. An operator strictly transmits if, $0 < i \leq q$.

$$C_1[i] = P_1[i] \text{ or } C_1[i] = P_2[i] \text{ and } C_2[i] = P_1[i] \text{ or } C_2[i] = P_2[i]$$

If one parent has blue eyes and brown hair and the other has brown eyes and red hair, the child must have either blue or brown eyes and either brown or red hair.

• Ergodicity. An operator is ergodic if, for any set of chromosomes, it is possible to access any point in the search space through the finite application of the operator. If the whole population has blue eyes, it must still be possible to produce a brown eyed child. This is a property that is desirable in a mutation operator.

• Properly Assort. An operator properly assorts if, given two instances of compatible o-schemata, it is possible for the operator to produce an offspring which is an instance of both o-schemata. i.e. for any subsets of elements of the two parents with no elements in contradictory positions, it is possible for the offspring to have both these subsequences. So, for example, if $P_1 = (1,2,3,4)$ and $P_2 = (3,2,4,1)$, it must be possible to have offspring $C_1 \in o$-schema $(1, *, 4, *)$ and $C_1 \in o$-schema $(*, 2, *, *)$ or any other compatible o-schemata for which the parents are instances. If one parent has blue eyes and the other brown hair it must be possible to recombine them to produce a child with blue eyes and brown hair as a result of the crossover. The only permutation operator we have found which properly assorts is Radcliffes
$R^3$ operator (random respectful recombination), which, as the name suggests, copies the common elements of both parents to the child and fills the remaining positions randomly. The $R^3$ operator is presented more as a useful starting point from which to design new operators than as a useful tool in it. The crossover operators for permutation problems described here are given in (Starkweather et al, 1991).

### 4.4.5 Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability: It is how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offsprings are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in the hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old populations to survive to next generation (Marek Obitko 1998a).

Mutation probability: It is how often parts of chromosome will be mutated. If there is no mutation, offsprings are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.
Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

4.5 STOPPING CONDITIONS FOR GENETIC ALGORITHM

The genetic algorithm uses the following five conditions to determine when to stop:

- **Generations**: The algorithm stops when the number of generation reaches the value of Generations.
- **Time limit**: The algorithm stops after running for an amount of time in seconds equal to Time limit.
- **Fitness limit**: The algorithm stops when the value of the fitness function for the best point in the current population is less than or equal to Fitness limit.
- **Stall generations**: The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length, Stall generations.
- **Stall time limit**: The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to stall time limit. The algorithm stops as soon as any one of these five conditions is met.

4.6 RELATED TECHNIQUES

- Genetic Programming (GP) is a related technique popularized by John Koza et al (2003) in which computer programs, rather than function parameters, are optimized. GP often uses tree-
based internal data structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.

- Interactive Genetic Algorithms (IGA) are genetic algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness function. For example, evolving images, music, artistic designs and forms to fit users' aesthetic preference. IGA has been implemented by Kim and Cho (2000).

- Simulated Annealing (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule. SA has been implemented by Dimovski and Gligoroski (2003b).

- Tabu Search (TS) is similar to SA in that both traverse the solution space by testing mutations of an individual solution. While SA generates only one mutated solution, Tabu Search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains
elements of the Tabu list, which is updated as the solution traverses the solution space (Andrew John Clark 1998).

- Ant Colony Optimization (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to GAs and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied. ACO has been implemented by Russell et al (2003a).

- Memetic Algorithm (MA), also called Hybrid Genetic Algorithm among others, is a relatively new evolutionary method where local search is applied during the evolutionary cycle. The idea of memetic algorithms comes from memes, which—unlike genes—can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms (Areibi et al 2001).

- Extremal Optimization (EO) Unlike GAs, which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure ("fitness"). The governing principle behind this algorithm is that of emergent improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions. EO has been implemented by Boettcher (2000).