GENETIC ALGORITHMS

Genetic algorithm is a *stochastic population-based search* method. Genetic Algorithms (GAs) are search algorithms based on the mechanics of the natural selection process (biological evolution). The most basic concept is that the strong tend to adapt and survive while the weak tend to die out. That is, optimization is based on evolution, and the "Survival of the fittest" concept.

GAs have the ability to create an initial population of feasible solutions, and then recombine them in a way to guide their search to only the most promising areas of the state space.

Each feasible solution is encoded as a chromosome (string) also called a genotype, and each chromosome is given a measure of fitness via a fitness (evaluation or objective) function.

_ The fitness of a chromosome determines its ability to survive and produce offspring.

_ A finite population of chromosomes is maintained.

_ GAs use probabilistic rules to evolve a population from one generation to the next. The generations of the new solutions are developed by genetic recombination operators:

- _ Biased Reproduction: selecting the fittest to reproduce
- _ Crossover: combining parent chromosomes to produce children chromosomes
- _ Mutation: altering some genes in a chromosome.

Most Important Parameters in GAs:

- _ Population Size
- _ Evaluation Function
- _ Crossover Method
- _ Mutation Rate

Use Genetic Algorithms:

_ When facing a problem of local minima.

_ When a good fitness function is available.

_ When a near-optimal, but not optimal solution is acceptable.

_ When the state-space is too large for other methods.

COMPONENTS OF GENETIC ALGORITHMS

The most important components in a GA consist of:

- Representation (definition of individuals)
- Evaluation function (or fitness function)
- Population

- Parent selection mechanism
- Variation operators (crossover and mutation)
- Survivor selection mechanism (replacement)

Representation

Objects forming possible solution within original problem context are called *phenotypes*, their encoding, the individuals within the GA, are called *genotypes*.

The representation step specifies the mapping from the phenotypes onto a set of genotypes.

Candidate solution, *phenotype* and *individual* are used to denote points of the space of possible solutions. This space is called *phenotype space*.

Chromosome, and *individual* can be used for points in the *genotye space*.

Elements of a chromosome are called *genes*. A value of a gene is called an *allele*.

The GA works with a coding of the parameter rather than the actual parameter. Depending on the particular problem, the coding may be Boolean-valued, integer-valued, real-valued, complex-valued, vector-valued, symbolic valued, or multiple-valued.

Evaluation function

This is just the cost or the objective function which represent what we want to maximize. For minimization problem use the reciprocal (i.e. 1/f).

Population

The role of the population is to hold possible solutions. A *population* is a multiset of genotypes. In GA, the population size is (almost always) constant. The bigger the size, the better the final solution, and the larger the execution time.

Parent Selection Mechanism

The role of *parent selection* (*mating selection*) is to distinguish among individuals based on their quality to allow the better individuals to become parents of the next generation.

Parent selection is *probabilistic*. Thus, high quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low quality individuals are often given a small, but positive chance; otherwise the whole search could become too greedy and get stuck in a local optimum.

Variation Operators

The role of variation operators is to create new individuals from old ones. Variation operators form the implementation of the elementary steps with the search space.

Mutation Operator

A unary variation operator is called *mutation*. It is applied to one genotype and delivers a modified *mutant*, the *child* or *offspring* of it.

In general, mutation is supposed to cause a random unbiased change.

Crossover Operator

A binary variation operator is called *recombination* or *crossover.* This operator merges information from two parent genotypes into one or two offspring genotypes.

Similarly to mutation, crossover is a stochastic operator: the choice of what parts of each parent are combined, and the way these parts are combined, depend on random drawings.

The principle behind crossover is simple: by mating two individuals with different but desirable features, we can produce an offspring which combines both of those features.

Survivor Selection Mechanism

The role of survivor selection is to distinguish among individuals based on their quality. In GA, the population size is (almost always) constant, thus a choice has to be made on which individuals will be allowed in the next generation. This decision is based on their fitness values, favoring those with higher quality.

As opposed to parent selection which is stochastic, survivor selection is often *deterministic*, for instance, ranking the unified multiset of parents and offspring and selecting the top segment (fitness biased), or selection only from the offspring (age-biased).

HOW DO GENETIC ALGORITHMS WORK ?

The details of how Genetic Algorithms work are explained below.

Initialization

Genetic algorithms are generally stated with an initial population that is *generated randomly*. Good selection of the initial population will give the GA a good start and speed up the evolutionary process.

Reproduction

There are two kinds of reproduction: generational reproduction and steady-state reproduction.

Generational Reproduction

In generational reproduction, the whole of a population is potentially replaced at each generation. The most often used procedure is to loop N/2 times, where N is the population size, select *two* chromosomes each time according to the current selection procedure, producing *two* children from those two parents, finally producing N new chromosomes.

Steady-state Reproduction

The steady-state method selects two chromosomes according to the current selection procedure, performs crossover on them to obtain one or two children, perhaps

applies mutation as well, and installs the result back into that population; the least fit of the population is destroyed.

Parent Selection mechanism

The effect of selection is to return a probabilistically selected parent. Although this selection procedure is stochastic, it does not imply GA employ a directionless search. The chance of each parent being selected is in some way related to its fitness.

Fitness-based selection

The standard, original method for parent selection is Roulette Wheel selection or fitness-based selection. In this kind of parent selection, each chromosome has a chance of selection that is directly proportional to its fitness. The effect of this depends on the range of fitness values in the current population.

Example: if fitness range from 5 to 10, then the fittest chromosome is twice as likely to be selected as a parent than the least fit.

If f_i is the fitness of individual i in the population, its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

where N is the number of individuals in the population

Rank-based selection

In the rank-based selection method, selection probabilities are based on a chromosome's relative rank or position in the population, rather than absolute fitness. So for a population of N solutions the best solution gets rank N, the second best rank N-1, etc.

Tournament-based selection

The original tournament selection is to choose K parents at random and returns the fittest one of these.

Crossover Operator

The crossover operator is the most important in GA. Crossover is a process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes. There are many kinds of crossover.

One-point Crossover

The procedure of one-point crossover is to randomly generate a number (less than or equal to the chromosome length) as the crossover position. Then, keep the bits before the number unchanged and swap the bits after the crossover position between the two parents.

Example: With the two parents selected above, we randomly generate a number 2 as the crossover position:

Parent 1: 7 3 7 6 1 3
Parent 2: 1 7 4 5 2 2
Then we get two children:
Child 1 : 7 3| 4 5 2 2
Child 2 : 1 7| 7 6 1 3

Parent 1	1 0 0	1 0 0 1 0 1 0
Parent 2	0 0 1	0 1 1 0 1 1 1
Child 1	1 0 0	0 1 1 0 1 1 1
Child 2	0 0 1	1 0 0 1 0 1 0

Two-point Cross Over

The procedure of two-point crossover is similar to that of one-point crossover except that we must select two positions and only the bits between the two positions are swapped. This crossover method can preserve the first and the last parts of a chromosome and just swap the middle part.
Example: With the two parents selected above, we randomly generate two numbers 2 and 4 as the crossover positions:
Parent1: 7 3 7 6 1 3
Parent2: 1 7 4 5 2 2
Then we get two children:
Child 1 : 7 3| 4 5| 1 3
Child 2 : 1 7| 7 6| 2 2

Parent 1	1 1 0 1 0 0	1 0 0 1	0 1 1
Parent 2	0 1 0 1 1 0	0 0 1 0	1 0 1
Child 1	1 1 0 1 0 0	0 0 1 0	0 1 1
Child 2	0 1 0 1 1 0	1 0 0 1	1 0 1

Uniform Crossover

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent.
Example: For each position, we randomly generate a number between 0 and 1, for example, 0.2, 0.7, 0.9, 0.4, 0.6, 0.1. If the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: 7 *3 *7 6 *1 3
Parent2: 1 *7 *4 5 *2 2
Then we get two children:
Child 1 : 7 7* 4* 6 2* 3
Child 2 : 1 3* 7* 5 1* 2

Inversion

Inversion operates as a kind of reordering technique. It operates on a single chromosome and inverts the order of the elements between two randomly chosen points on the chromosome. While this operator was inspired by a biological process, it requires additional overhead.

Example: Given a chromosome 3 8 4 8 6 7. If we randomly choose two positions 2, 5 and apply the inversion operator, then we get the new string: 3 6 8 4 8 7.

Mutation

Mutation has the effect of ensuring that all possible chromosomes are reachable. With crossover and even inversion, the search is constrained to alleles which exist in the initial population. The mutation operator can overcome this by simply randomly selecting any bit position in a string and changing it. This is useful since crossover and inversion may not be able to produce new alleles if they do not appear in the initial generation.

Example: Assume that we have already used crossover to get a new string: 7 3 4 5 1 3. Assume the mutation rate is 0.001 (usually a small value). Next, for the first bit 7, we generate randomly a number between 0 and 1. If the number is less than the mutation rate (0.001), then the first bit 7 needs to mutate. We generate another number between 1 and the maximum value 8, and get a number (for example 2). Now the first bit mutates to 2. We repeat the same procedure for the other bits. In our example, if only the first bit mutates, and the rest of the bits don't mutate, then we will get a new chromosome as below:

2 3 4 5 1 3

CONSTRAINT HANDLING IN GENETIC ALGORITHMS

There are many ways to handle constraints in a GA. At the high conceptual level we can distinguish two cases: indirect constraint handling and direct constraint handling.

Indirect constraint handling means that we circumvent the problem of satisfying constraints by incorporating them in the fitness function *f* such that *f* optimal implies that the constraints are satisfied, and use the power of GA to find a solution.

Direct constraint handling means that we leave the constraints as they are and 'adapt' the GA to enforce them.

Notice that direct and indirect constraint handling can be applied in combination, i.e., in one application we can handle some constraints directly and others indirectly.

Formally, indirect constraint handling means transforming constraints into optimization objectives.

Direct constraint handling

Treating constraints directly implies that violating them is not reflected in the fitness function, thus there is no bias towards chromosomes satisfying them. Therefore, the population will not become less and less infeasible w.r.t. these constraints. This means that we have to create and maintains feasible chromosomes in the population. The basic problem in this case is that the regular operators are blind to constraints, mutating one or crossing over two feasible chromosomes can result in infeasible offspring. Typical approaches to handle constraints directly are the following:

- eliminating infeasible candidates
- repairing infeasible candidates
- preserving feasibility by special operators
- decoding, i.e. transforming the search space.

Eliminating infeasible candidates is very inefficient, and therefore hardly applicable. Repairing infeasible candidates requires a repair procedure that modifies a given chromosome such that it will not violate constraints. This technique is thus problem dependent.

The preserving approach amounts to designing and applying problem-specific operators that do preserve the feasibility of parent chromosomes. Note that the preserving approach requires the creation of a feasible initial population, which can be NP-complete.

Decoding can simplify the problem search space and allow an efficient genetic algorithm. Formally, decoding can be seen as shifting to a search space that is different from the Cartesian product of the domains of the variables in the original problem formulation.

Indirect Constraint Handling

In the case of indirect constraint handling the optimization objectives replacing the constraints are viewed *penalties* for constraint violation hence to be minimized. In general penalties are given for violated constraints although some GAs allocate penalties for wrongly instantiated variables or as the distance to a feasible solution.

Advantages of indirect constraint handling are:

- generality
- reduction of the problem to ‘simple’ optimization
- possibility of embedding user preferences by means of weights.

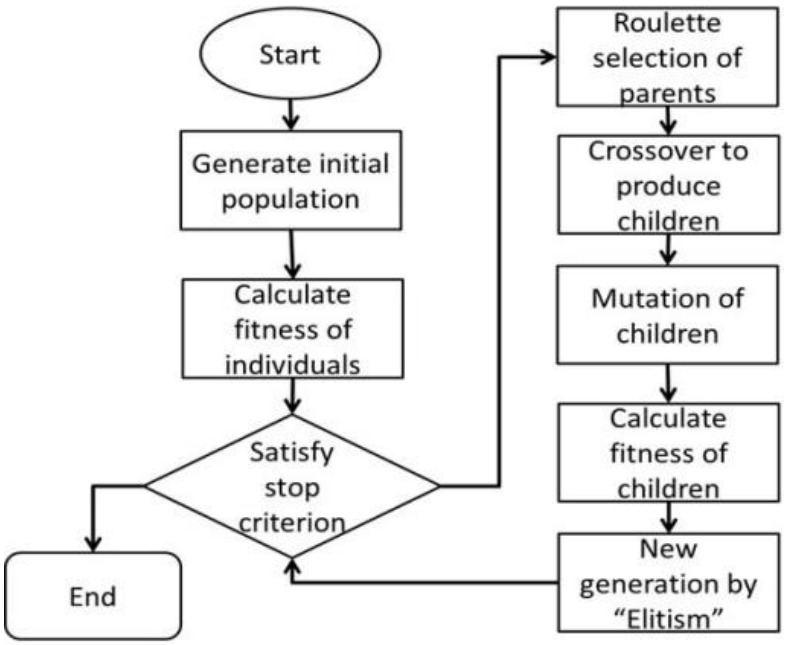
Disadvantages of indirect constraint handling are:

- loss of information by packing everything in a single number
- does not work well with sparse problems.

GENETIC AGORITHM

The general scheme of a GA can be given as follows:

begin
  INITIALIZE population with random candidate solutions;
  EVALUATE each candidate;
  repeat
    SELECT parents;
    RECOMBINE pairs of parents;
    MUTATE the resulting children;
    EVALUATE children;
    SELECT individuals for the next generation
  until TERMINATION-CONDITION is satisfied
end



GA flow chart

Matlab Roulette Wheel code

```
function choice = Roulette_Wheel (weights)% ex: weights=[1 5 3 15 8 1]
  accumulation = cumsum(weights); % =[ 1 6 9 24 32 33]
  p = rand() * accumulation(end);
  chosen_index = -1;
  for index = 1 : length(accumulation)
    if (accumulation(index) > p)
      chosen_index = index;
      break;
    end
  end
  choice = chosen_index;
```