SOFT COMPUTING TECHNIQUES

UNIT-4

UNIT-IV:

Genetic Algorithm: Introduction to Genetic algorithms, Genetic algorithms, procedures of GA's, working of GA's, Travelling sales man problem, Evolutionary programming, working principle of GA Machine learning classifier system

Genetic algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

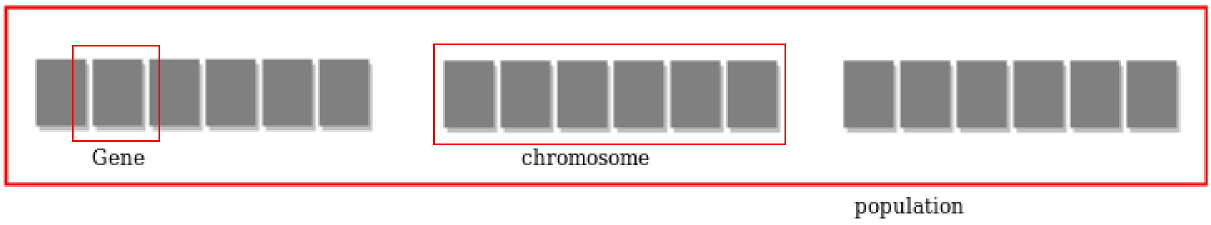
Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosomes of the population. Following is the foundation of GAs based on this analogy –

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

A Fitness Score is given to each individual which shows the ability of an individual to “compete”. The individual having optimal fitness score (or near optimal) are sought.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce better offspring by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.

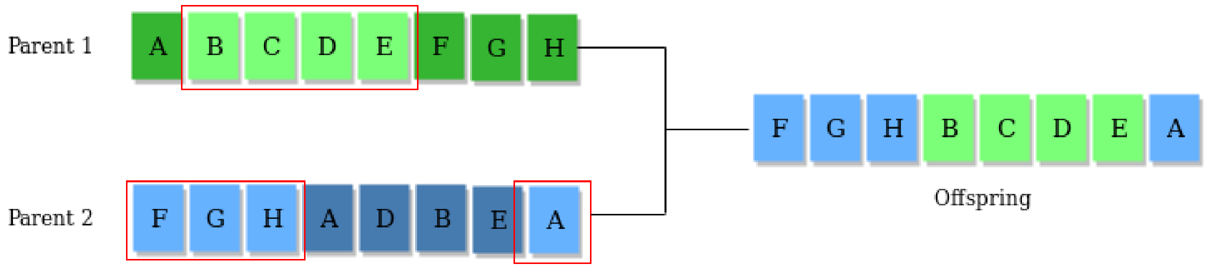
Each new generation has on average more “better genes” than the individual (solution) of previous generations. Thus each new generations have better “partial solutions” than previous generations. Once the offspring produced having no significant difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

Operators of Genetic Algorithms

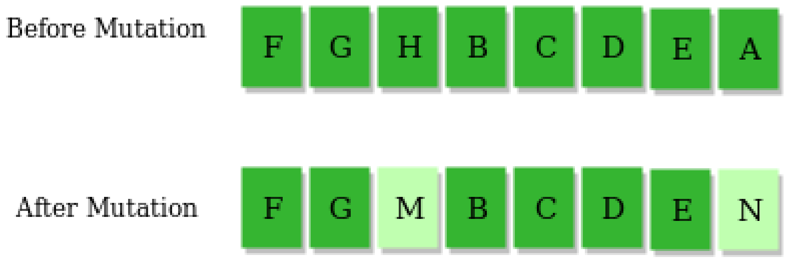
Once the initial generation is created, the algorithm evolves the generation using following operators –

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For example –



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For example –



The whole algorithm can be summarized as –

1) Randomly initialize populations p

2) Determine fitness of population 3) Until convergence repeat:

a) Select parents from population

b) Crossover and generate new population

c) Perform mutation on new population

d) Calculate fitness for new population

Application of Genetic Algorithms

- Recurrent Neural Network
- Mutation testing
- Code breaking

- Filtering and signal processing
- Learning fuzzy rule base

Procedures of GA

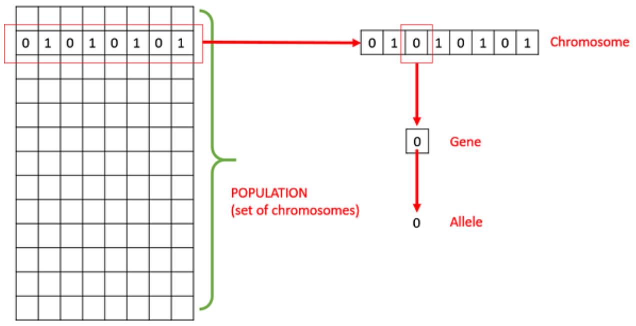
Genetic Algorithms – Fundamentals

This section introduces the basic terminology required to understand GAs. Also, a generic structure of GAs is presented in both pseudo-code and graphical forms. The reader is advised to properly understand all the concepts introduced in this section and keep them in mind when reading other sections of this tutorial as well.

Basic Terminology

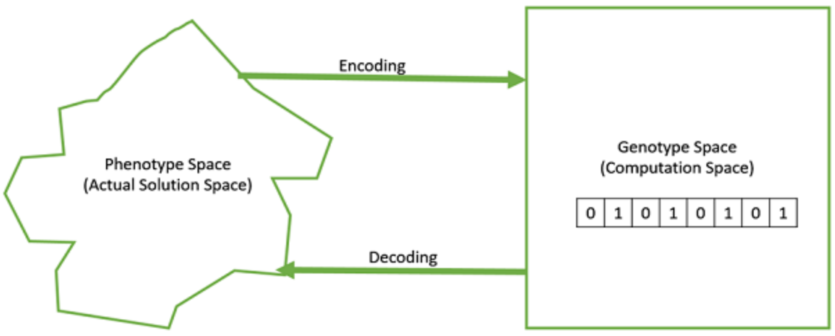
Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.

- Population – It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- Chromosomes – A chromosome is one such solution to the given problem.
- Gene – A gene is one element position of a chromosome.
- Allele – It is the value a gene takes for a particular chromosome.



- Genotype – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- Phenotype – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.

- Decoding and Encoding − For simple problems, the phenotype and genotype spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.
- For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.
- However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A 0 at position x represents that xth item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

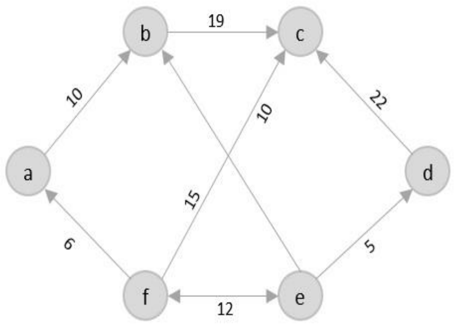


- Fitness Function − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- Genetic Operators − These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

Travelling sales man problem

The travelling salesman problem is a graph computational problem where the salesman needs to visit all cities (represented using nodes in a graph) in a list just once and the distances (represented using edges in the graph) between all these cities are known. The solution that is needed to be found for this problem is the shortest possible route in which the salesman visits all the cities and returns to the origin city.

If you look at the graph below, considering that the salesman starts from the vertex ‘a’, they need to travel through all the remaining vertices b, c, d, e, f and get back to ‘a’ while making sure that the cost taken is minimum.



here are various approaches to find the solution to the travelling salesman problem: naive approach, greedy approach, dynamic programming approach, etc. In this tutorial we will be learning about solving travelling salesman problem using greedy approach.

Travelling Salesperson Algorithm

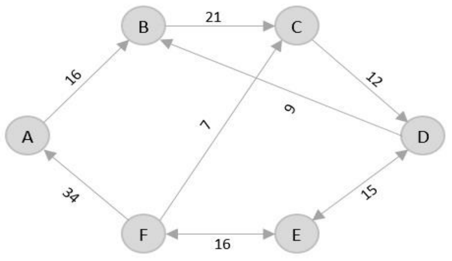
As the definition for greedy approach states, we need to find the best optimal solution locally to figure out the global optimal solution. The inputs taken by the algorithm are the graph G {V, E}, where V is the set of vertices and E is the set of edges. The shortest path of graph G starting from one vertex returning to the same vertex is obtained as the output.

Algorithm

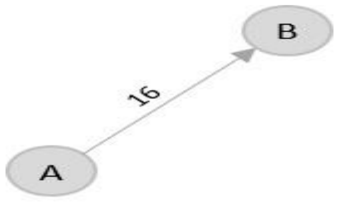
- Travelling salesman problem takes a graph G {V, E} as an input and declare another graph as the output (say G') which will record the path the salesman is going to take from one node to another.
- The algorithm begins by sorting all the edges in the input graph G from the least distance to the largest distance.
- The first edge selected is the edge with least distance, and one of the two vertices (say A and B) being the origin node (say A).
- Then among the adjacent edges of the node other than the origin node (B), find the least cost edge and add it onto the output graph.
- Continue the process with further nodes making sure there are no cycles in the output graph and the path reaches back to the origin node A.
- However, if the origin is mentioned in the given problem, then the solution must always start from that node only. Let us look at some example problems to understand this better.

Examples

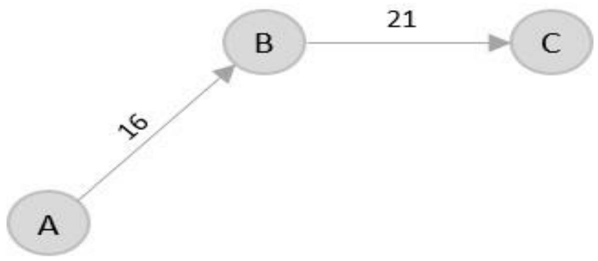
Consider the following graph with six cities and the distances between them –



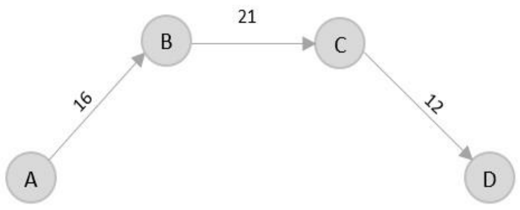
From the given graph, since the origin is already mentioned, the solution must always start from that node. Among the edges leading from A, A → B has the shortest distance.



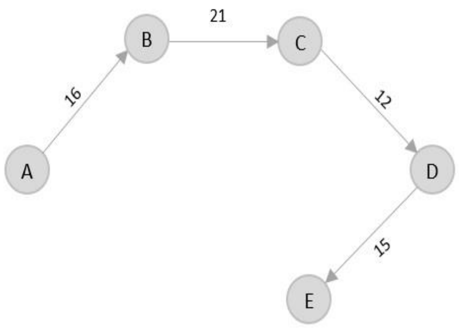
Then, B → C has the shortest and only edge between, therefore it is included in the output graph.



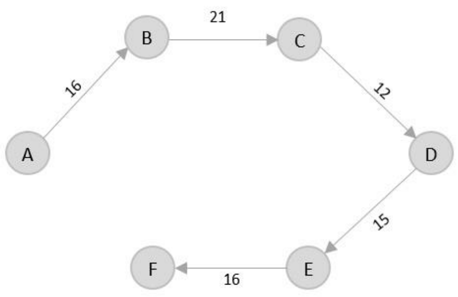
There’s only one edge between C → D, therefore it is added to the output graph.



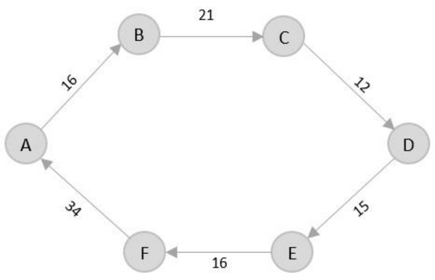
There’s two outward edges from D. Even though, D → B has lower distance than D → E, B is already visited once and it would form a cycle if added to the output graph. Therefore, D → E is added into the output graph.



There’s only one edge from e, that is E → F. Therefore, it is added into the output graph.



Again, even though F → C has lower distance than F → A, F → A is added into the output graph in order to avoid the cycle that would form and C is already visited once.



The shortest path that originates and ends at A is A → B → C → D → E → F → A
The cost of the path is: 16 + 21 + 12 + 15 + 16 + 34 = 114.
Even though, the cost of path could be decreased if it originates from other nodes but the question is not raised with respect to that.

Evolutionary programming

An evolutionary algorithm is evolutionary AI-based computer software that solves issues by employing processes that mimic the behaviors of living things. As such, it needs mechanisms that are generally related to biological evolution, including reproduction, mutation, and recombination.

An example of data extraction and transformation tools is the ETL-EXTRACT tool suite from evolutionary technologies. Extraction is the service of extracting information from a source system for additional help in a data warehouse environment. It is the first procedure of the ETL process. After the extraction, this data can be changed and loaded into the data warehouse.

The process of data extraction contains the retrieval of information from messy data sources. The data extracts are loaded into the staging operation of the relational database. Hence extraction logic is utilized and the source system is asked for data using software programming interfaces.

This product automates and expedites the migration of data between dissimilar storage environments, potentially allowing organizations to save up to 93% of time and cost. It enables users to –

- It can populate and maintain data warehouses.
- It is used to integrate disparate systems.
- It can migrate data to new databases, applications, and platforms.
- It can move to new architectures, such as distributed client-server.

The ETL-EXTRACT tool suite is a flexible solution that –

- It can support data collection, conversion, and migration from a variety of platforms, OS, and DBMS to any other.
- It can be automatically generated and executes programs in appropriate languages for source and target platforms.
- It provides a metadata facility that allows users to track information about stored data.
- It provides a graphical interface that allows users to indicate how to move data, through a simple point and click interaction.

Components of ETL-EXTRACT tool suite –

- The ETL-EXTRACT executive.
- The ETL-EXTRACT workset browser.
- The ETL-EXTRACT metadata facility.
- The metadata store database.
- The metadata exchange library.

The key feature of this tool is its ability to support conditional data selection and transformation that can be done programmatically. It also provides a broad range of metadata management options. The ETL-EXTRACT tool suite consists of two sets of productivity tools –

Master toolset − The environment editor allows the specification of each different platform and system OS.

The schema editor provides easy access for browsing or updating schema information.

The grammar editor offers a simplified means for defining customized condition

The template editor enables rapid specification of programming rules to shape the way data retrieval, conversion, and populate programs.

Conversion editor − It allows users to –

- It is used to create programs that test and transform data.
- It can merge data from multiple systems to generate a new database.
- It can be used to retrieve data from one or more databases or file formats.
- It can populate any number of DBMS or file formats.