EXPERIMENT NO: 8

8. Implement an algorithm to demonstrate the significance of geneticalgorithm

Aim:

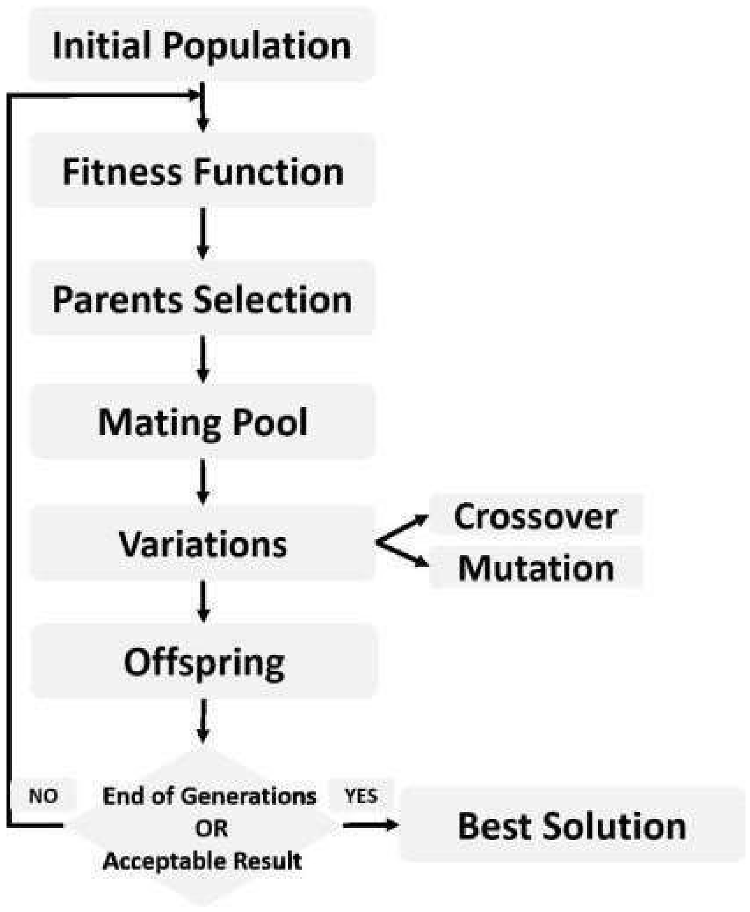
To implement an algorithm to demonstrate the significance of genetic algorithm

Theory:

Genetic algorithm

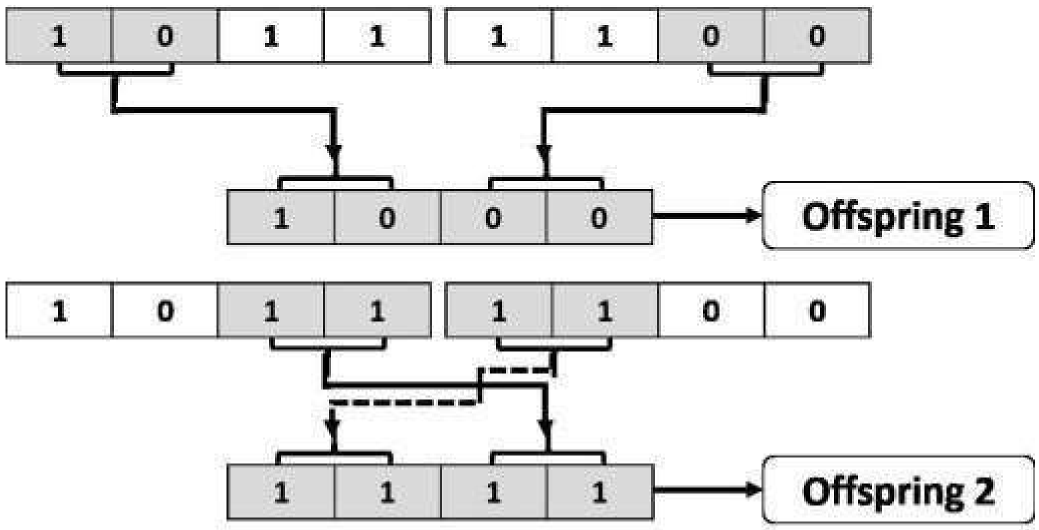
The genetic algorithm is a population-based evolutionary algorithm, where a group of solutions works together to find the optimal parameters for a problem. The belowfigure, from this book, summarizes all the steps in the genetic algorithm.

The population of solutions is initialized randomly, where each solution consists of anumber of genes. The quality of solutions is assessed using a fitness function, whichreturns a numeric value representing how fit the solution is.



The high-quality (high-fitness) solutions survive longer than the ones with low fitness. The higher the fitness, the higher probability of selecting the solution as a parent to produce new offspring. To produce the offspring, pairs of parents mate using the crossover operation, where a new solution is generated that carries genesfrom its parents.

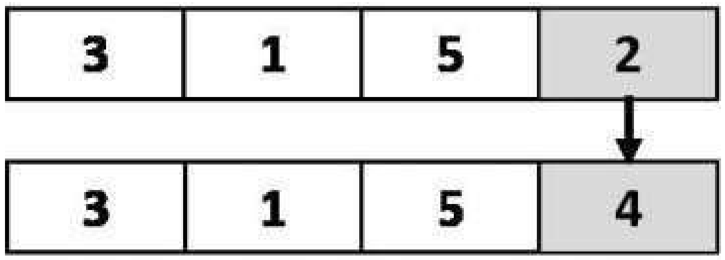
After crossover, mutation is applied to add some random changes over the solution.The evolution continues through a number of generations to reach the highest- quality solution.



If there are some bad genes within the parents, they will definitely be transferred totheir children after crossover. The mutation operation plays a crucial role in fixing this issue.

During mutation, some genes are randomly selected from each child where some random changes are applied. Genes are selected based on a random probability for each gene. If the probability of mutating a gene is smaller than or equal to a predefined threshold, then this gene will be selected for mutation. Otherwise, it will be skipped. We‟ll discuss mutation probability later on.

Let‟s assume there are 4 genes in the solution, as in the next figure, where only thelast gene is selected for mutation. A random change is applied to change its old value **2** and the new value is **4**



Source Code:

```
# Python3 program to create target string, starting from#
random string using Genetic Algorithm

import random

# Number of individuals in each generation
POPULATION_SIZE = 100

# Valid genes
GENES =
'''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''

# Target string to be generated
TARGET = "CMR College"

  class
    Individual(object):'''
    Class representing individual in population'''
    def___init_(self, chromosome):
            self.chromosome = chromosome
            self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):'''
            create random genes for mutation'''
            global GENES
            gene = random.choice(GENES)
            return gene

    @classmethod
    def create_gnome(self):'''
            create chromosome or string of genes'''
            global TARGET
            gnome_len = len(TARGET)
            return [self.mutated_genes() for _ in range(gnome_len)]

    def mate(self, par2):
            '''
            Perform mating and produce new offspring'''

            # chromosome for offspring
            child_chromosome = []
            for gp1, gp2 in zip(self.chromosome, par2.chromosome):
                    # random probability prob =
                    random.random()

                    # if prob is less than 0.45, insert gene# from
                    parent 1
                    if prob < 0.45:
                            child_chromosome.append(gp1)

                    # if prob is between 0.45 and 0.90, insert# gene
                    from parent 2
                    elif prob < 0.90:
                            child_chromosome.append(gp2)

                    # otherwise insert random gene(mutate),# for
                    maintaining diversity
                    else:
                            child_chromosome.append(self.mutated_genes())

            # create new Individual(offspring) using#
            generated chromosome for offspring return
            Individual(child_chromosome)

    def cal_fitness(self):
            '''
            Calculate fitness score, it is the number of
            characters in string which differ from targetstring.
            '''
            global TARGET
            fitness = 0
            for gs, gt in zip(self.chromosome, TARGET):if gs
                    != gt: fitness+= 1
            return fitness

    # Driver code
  def main():
    global POPULATION_SIZE

    #current generation
    generation = 1

    found = False
    population = []

    # create initial population
    for _ in range(POPULATION_SIZE):
                            gnome = Individual.create_gnome() population.append(Individual(gnome))
    while not found:

            # sort the population in increasing order of fitness score population
            = sorted(population, key = lambda x:x.fitness)
```

```
        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target# and
        break the loop
        if population[0].fitness <= 0:
                found = True
                break

        # Otherwise generate new offsprings for new generation
        new_generation = []

        # Perform Elitism, that mean 10% of fittest population# goes to
        the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        # From 50% of fittest population, Individuals# will
        mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
                parent1 = random.choice(population[:50])parent2
                = random.choice(population[:50])child =
                parent1.mate(parent2)
                new_generation.append(child)

        population = new_generation

        print("Generation: {}\tString: {}\tFitness: {}".\
                format(generation,
                "".join(population[0].chromosome),
                population[0].fitness))

        generation += 1


    print("Generation: {}\tString: {}\tFitness: {}".\
        format(generation,
        "".join(population[0].chromosome),
        population[0].fitness))

    if __name__ == '__main__':
        main()
```

Output:

```
Generation: 1      String: CG H      Fitness: 3
Generation: 2      String: CG H      Fitness: 3
Generation: 3      String: CG H      Fitness: 3
Generation: 4      String: APR       Fitness: 2
Generation: 5      String: APR       Fitness: 2
Generation: 6      String: CMRS      Fitness: 1
Generation: 7      String: CMRS      Fitness: 1
Generation: 8      String: CMR       Fitness: 0
```

Sample Viva Questions:

1. Explain some basic concepts and terms related to *Genetic Algorithm*

- **Population:** This is a subset of all the *probable* solutions that can solve the given problem.
- **Chromosomes:** A chromosome is one of the solutions in the population.
- **Gene:** This is an element in a chromosome.
- **Allele:** This is the value given to a gene in a specific chromosome.
- **Fitness function:** This is a function that uses a specific *input* to produce an improved *output*. The solution is used as the input while the output is in the form of solution suitability.

2. What is a *Mutation* and why is it programmed into the algorithm?

- ***Mutation*** introduces new patterns in the chromosomes, and it helps to find solutions inuncharted areas.
- ***Mutations*** are implemented as *random* changes in the chromosomes. It may be programmed, for example, as a *bit flip* where a single bit of the chromosome changes.
- The purpose of *mutation* is to periodically *refresh* the population.

3. Compare the *Single-Point* and *Two-Point* crossover

- In **Single-point Crossover**, a location is selected *randomly* on both the parents. The location is the same in both the parents *chromosome*. The genes to the right of the *crossover point* is swapped between the two parents.

- In **Two-point Crossover**, *two* locations are selected *randomly* in the parents' chromosome. Then the genes in between the two points are swapped between the twoparents.

4. **Name some *types* of *Mutation* in GA**

- **Flip bit mutation:** This mutation can be applied to binary chromosomes. In this type of mutation, a single bit is selected randomly and it is *flipped*. It can be programmed so that multiple bits are flipped instead of one.
- **Swap mutation:** When this mutation is applied, two *genes* are randomly selected andtheir values are swapped within the same chromosome.
- **Inversion mutation:** In this mutation, *random* sequences of genes are selected and theorder of those genes is *reversed*.
- **Scramble mutation:** In this mutation, a *random* sequence of genes is selected and thegenes in that sequence is *shuffled*.

5. **What are some *Stopping Conditions* that a genetic algorithm may implement?Some stopping conditions that can be implemented are:**

- Stopping when the **maximum (absolute) number of generations** has been reached.This limits the resources and time required by the algorithm.
- Stopping if there is **no noticeable improvement in the fitness score**. This can be implemented by storing the best fitness score achieved in every generation and comparingthe current best value to the one achieved at every generation. If the difference is small compared to the threshold, then the algorithm can stop.
- Stopping after a **predetermined time** after starting the computation..

EXPERIMENT NO:9

9. Implement the finite words classification system using Back-propagationalgorithm

Aim:

To implement the finite words classification system using Back-propagation algorithm

Theory:

Artificial neural networks (ANNs) provide a general, practical method for learningrealvalued, discrete-valued, and vector-valued functions from examples.

Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.

ANN learning is robust to errors in the training data and has been successfully appliedto problems such as interpreting visual scenes, speech recognition, and learning robotcontrol.

Source Code:

```
import numpy as np # numpy is

X = np.array(([2, 9], [1, 5], [3, 6]),dtype=float)

y = np.array(([92], [86], [89]),dtype=float)

X = X/np.amax(X,axis=0) # Normalizey
      = y/100
def sigmoid(x):

return 1/(1+np.exp(-x))def
      sigmoid_grad(x):
return x * (1 - x)

# Variable initialization

epoch=1000 #Setting training iterationseta
      =0.2 #Setting learning rate (eta)
input_neurons = 2 #number of features in data set

hidden_neurons = 3 #number of hidden layers neurons
```

```
output_neurons = 1 #number of

# Weight and bias - Random initialization
      wh=np.random.uniform(size=(input_neurons,hidden_neurons)) # 2x3
      bh=np.random.uniform(size=(1,hidden_neurons)) # 1x3
      wout=np.random.uniform(size=(hidden_neurons,output_neurons))
      bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):

h_ip=np.dot(X,wh) + bh # Dot product + biash_act =
      sigmoid(h_ip) # Activation function
      o_ip=np.dot(h_act,wout) + bout
output = sigmoid(o_ip)
      #Backpropagation
# Error at Output layer

Eo = y-output # Error at o/p
      outgrad =
      sigmoid_grad(output)
d_output = Eo* outgrad # Errj=Oj(1-Oj)(Tj-Oj)

Eh = d_output.dot(wout.T) # .T means transpose

hiddengrad = sigmoid_grad(h_act) # How much hidden layer wts contributed to errord_hidden = Eh *
      hiddengrad
wout += h_act.T.dot(d_output) *eta # Dotproduct of nextlayererror and currentlayeropwh +=
      X.T.dot(d_hidden) *eta
print("Normalized Input: \n" + str(X))
      print("Actual Output: \n" + str(y))
      print("Predicted Output: \n" ,output
```

Output:

```
Normalized Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.77772836]
 [0.76211175]
 [0.77933662]]
```

Sample Viva Questions:

1. **How do you implement the back propagation algorithm?**

Backpropagation Algorithm:
Step 1: Inputs X, arrive through the preconnected path.
Step 2: The input is modeled using true weights W. Weights are usually chosenrandomly.
Step 3: Calculate the output of each neuron from the input layer to the hidden layer tothe output layer.

2. **How backpropagation algorithm can be used in classification?**

Static backpropagation networks can solve static classification problems, such as opticalcharacter recognition (OCR). Recurrent backpropagation. The recurrent backpropagationnetwork is used for fixed-point learning. Recurrent backpropagation activation feeds forward until it reaches a fixed value.

3. **What is the advantage of back propagation algorithm?**

Most prominent advantages of Backpropagation are: Backpropagation is fast, simple andeasy to program. It has no parameters to tune apart from the numbers of input. It is a flexible method as it does not require prior knowledge about the network.

4. **Why is it called backpropagation?**

The calculation of the error δ j k \delta_j^{k} δjk will be shown to be dependent on the values of error terms in the next layer. Thus, computation of the error terms will proceedbackwards from the output layer down to the input layer. This is where backpropagation,or backwards propagation of errors, gets its name.

5. **What is the difference between back propagation and guided back propagation**?

"Guided Backpropagation" uses vanilla backpropagation, except at the ReLUs. Guided Backpropagation combines vanilla backpropagation at ReLUs (leveraging which elementsare positive in the preceding feature map) with DeconvNets (keeping only positive error signals.