VIDYA JETHWA
ROLL NO: 529

DML

ASSIGNMENT

AIM: IMPLEMENT GENETIC ALGORITHM WITH THREE OPERATORS

Explanation:

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.

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.

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).

VIDYA JETHWA
ROLL NO: 529

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

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
```

VIDYA JETHWA
ROLL NO: 529

CODE:

```
import random
POPULATION_SIZE = 100

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

TARGET = "Hello World"

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
            '''

            child_chromosome = []
```

```
            for gp1, gp2 in zip(self.chromosome,
par2.chromosome):


                prob = random.random()


                if prob < 0.45:
                    child_chromosome.append(gp1)


                elif prob < 0.90:
                    child_chromosome.append(gp2)


                else:

        child_chromosome.append(self.mutated_genes())


            return Individual(child_chromosome)


        def cal_fitness(self):
            '''
            Calculate fitness score, it is the number of
            characters in string which differ from target
```

```
            string.
            '''
            global TARGET
            fitness = 0
            for gs, gt in zip(self.chromosome, TARGET):
                  if gs != gt: fitness+= 1
            return fitness


def main():
      global POPULATION_SIZE

      generation = 1

      found = False
      population = []

      for _ in range(POPULATION_SIZE):
                        gnome = Individual.create_gnome()
                        population.append(Individual(gnome))

      while not found:
```

```
        population = sorted(population, key = lambda
x:x.fitness)

        if population[0].fitness <= 0:
            found = True
            break

        new_generation = []

        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        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()
```

VIDYA JETHWA
ROLL NO: 529

OUTPUT: