AISC

Class: B.E COMP

Experiment 04: Development optimization solution for a problem using Genetic Algorithm

Learning Objective:

Basic Experiments

Solve a given problem using Genetic Algorithm

Tools: Python

Theory:

Genetic Algorithm (GA)

The genetic algorithm is a random-based classical evolutionary algorithm. By random here we mean that in order to find a solution using the GA, random changes applied to the current solutions to generate new ones. Note that GA may be called Simple GA (SGA) due to its simplicity compared to other EAs.

GA is based on Darwin's theory of evolution. It is a slow gradual process that works by making changes to the making slight and slow changes. Also, GA makes slight changes to its solutions slowly until getting the best solution.

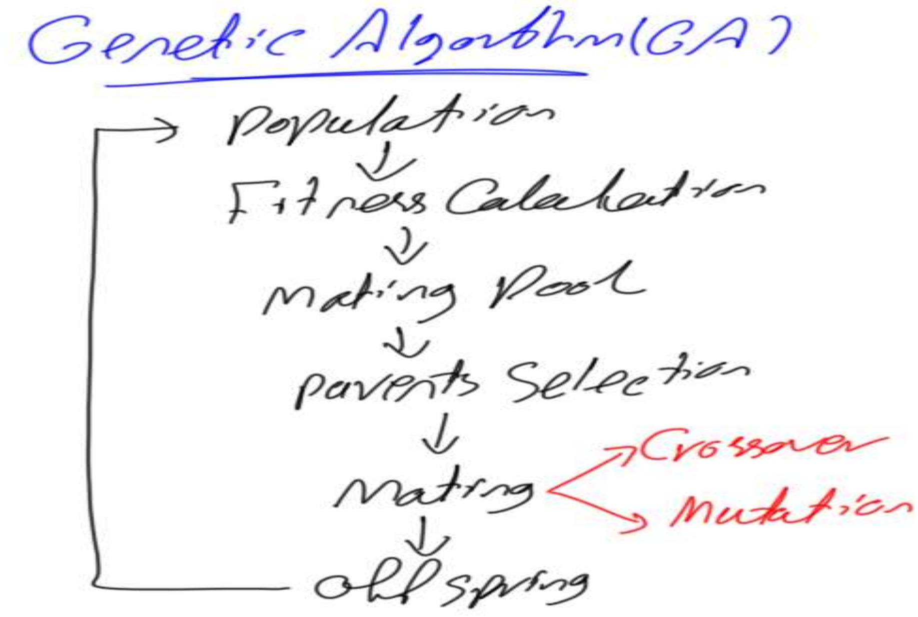
Here is the description of how the GA works:

GA works on a population consisting of some solutions where the population size (popsize) is the number of solutions. Each solution is called individual. Each individual solution has a chromosome. The chromosome is represented as a set of parameters (features) that defines the individual. Each chromosome has a set of genes. Each gene is represented by somehow such as being represented as a string of 0s and 1s

lso, each individual has a fitness value. To select the best individuals, a fitness function is used. The result of the fitness function is the fitness value representing the quality of the solution. The higher the fitness value the higher the quality the solution. Selection of the best individuals based on their quality is applied to generate what is called a mating pool where the higher quality individual has higher probability of being selected in the mating pool.

The individuals in the mating pool are called parents. Every two parents selected from the mating pool will generate two offspring (children). By just mating high-quality individuals, it is expected to get a better quality offspring than its parents. This will kill the bad individuals from generating more bad individuals. By keeping selecting and mating high-quality individuals, there will be higher chances to just keep good properties of the individuals and leave out bad ones. Finally, this will end up with the desired optimal or acceptable solution.

But the offspring currently generated using the selected parents just have the characteristics of its parents and no more without changes. There is no new added to it and thus the same drawbacks in its parents will actually exist in the new offspring. To overcome such problem, some changes will be applied to each offspring to create new individuals. The set of all newly generated individuals will be the new population that replaces the previously used old population. Each population created is called a generation. The process of replacing the old population by the new one is called replacement. Figure 2 summarizes the steps of GA.



There are two questions to be answered to get the full idea about GA:

1. How the two offspring are generated from the two parents?
2. How each offspring gets slightly changed to be an individual?

We will answer these questions later.

Chromosome Representation and Evaluation

There are different representations available for the chromosome and the selection of the proper representation is problem specific. The good representation is what makes the search space smaller and thus easier search.

The representations available for the chromosome including:

· **Binary**: Each chromosome is represented as a string of zeros and ones.

· **Permutation**: Useful for ordering problems such as travelling salesman problem.

· **Value**: The actual value is encoded as it is.

For example, if we are to encode the number 7 in binary, it might look as shown in figure 3.

3	2	1	0
0	1	1	1

Each part of the above chromosome is called gene. Each gene has two properties. The first one is its value (allele) and the second one is the location (locus) within the chromosome which is the number above its value.

Each chromosome has two representations.

1. **genotype**: The set of genes representing the chromosome.
2. **phenotype**: The actual physical representation of the chromosome.

In the above example, binary of 0111 is the genotype and 7 is the phenotype representation.

After representing each chromosome the right way to serve to search the space, next is to calculate the fitness value of each individual. Assume that the fitness function used in our example is:
$f(x)=2x+2$
Where x is the chromosome value
Then the fitness value of the previous chromosome is:
$f(7)=2(7)+2=16$
The process of calculating the fitness value of a chromosome is called evaluation.

Initialization

After getting how to represent each individual, next is to initialize the population by selecting the proper number of individuals within it.

Selection

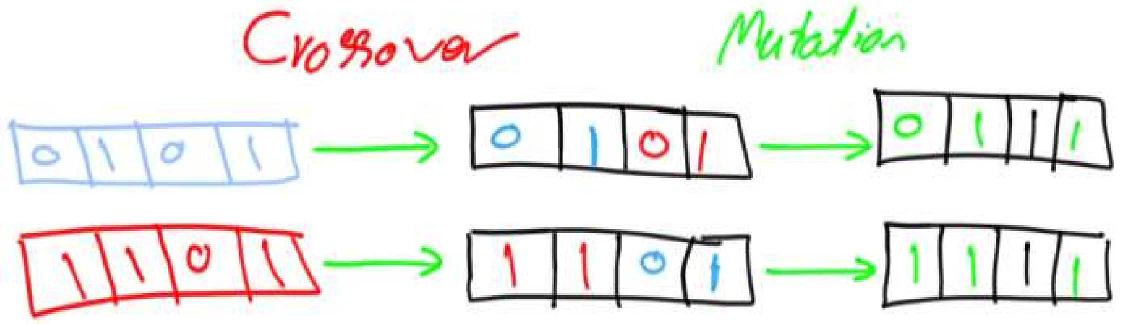
Next is to select a number of individuals from the population in the mating pool. Based on the previously calculated fitness value, the best individuals based on a threshold are selected. After that step, we will end selecting a subset of the population in the mating pool.

Variation Operators

Based on the selected individuals in the mating pool, parents are selected for mating. The selection of each two parents may be by selecting parents sequentially (1–2, 3–4, and so on). Another way is random selection of the parents.

For every two parents selected, there are a number of variation operators to get applied such as:

1. Crossover (recombination)
2. Mutation



Crossover

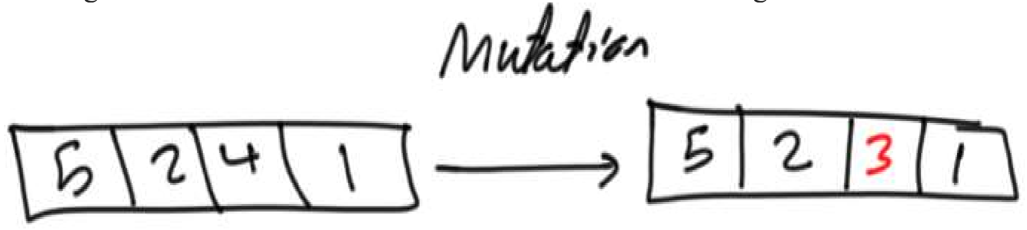
Crossover in GA generates new generation the same as natural mutation. By mutating the old generation parents, the new generation offspring comes by carrying genes from both parents. The amount of genes carried from each parent is random. Remember that GA is random-based EA. Sometimes the offspring takes half of its genes from one parent and the other half from the other parent and sometimes such percent changes. For every two parents, crossover takes place by selecting a random point in the chromosome and exchanging genes before and after such point from its parents. The resulting chromosomes are offspring. Thus operator is called single-point crossover.

Note that crossover is important and without it, the offspring will be identical to its parent.

Mutation

Next variation operator is mutation. For each offspring, select some genes and change its value. Mutation varies based on the chromosome representation but it is up to you to decide how to apply mutation. If the encoding is binary (i.e. the value space of each gene have just two values 0 and 1), then flip the bit value of one or more genes.

But if the gene value comes from a space of more than two values such as 1,2,3,4, and 5, then the binary mutation will not be applicable and we should find another way. One way is by selecting a random value from such set of values as shown in figure 5.



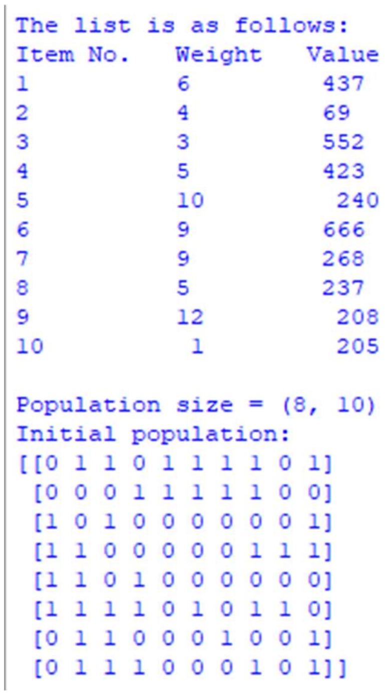
Note that without mutation the offspring will have all of its properties from its parents. To add new features to such offspring, mutation took place. But because mutation occurs randomly, it is not recommended to increase the number of genes to be applied to mutation. The individual after mutation is called mutant.

Implimentation:

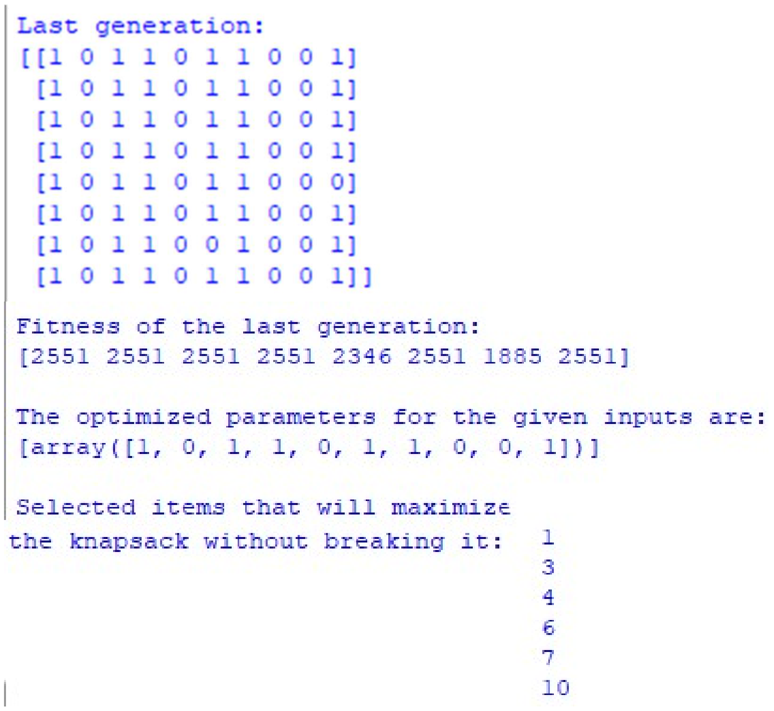
SOURCE CODE:

```
import numpy as np
import pandas as pd
import random as rd
from random import randint
import matplotlib.pyplot as plt
item_number = np.arange(1,11)
weight = np.random.randint(1, 15, size = 10)
value = np.random.randint(10, 750, size = 10)
knapsack_threshold = 35    #Maximum weight that the bag of thief can hold
print('The list is as follows:')
print('Item No.   Weight   Value')
for i in range(item_number.shape[0]):
    print('{0}          {1}         {2}'.format(item_number[i], weight[i], value[i]))

    solutions_per_pop = 8
pop_size = (solutions_per_pop, item_number.shape[0])
print('\nPopulation size = {}'.format(pop_size))
initial_population = np.random.randint(2, size = pop_size)
initial_population = initial_population.astype(int)
num_generations = 50
print('Initial population: \n{}'.format(initial_population))

def cal_fitness(weight, value, population, threshold):
    fitness = np.empty(population.shape[0])
    for i in range(population.shape[0]):
        S1 = np.sum(population[i] * value)
        S2 = np.sum(population[i] * weight)
        if S2 <= threshold:
            fitness[i] = S1
        else :
            fitness[i] = 0
    return fitness.astype(int)

def selection(fitness, num_parents, population):
    fitness = list(fitness)
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        max_fitness_idx = np.where(fitness == np.max(fitness))
        parents[i,:] = population[max_fitness_idx[0][0], :]
        fitness[max_fitness_idx[0][0]] = -999999
    return parents

def crossover(parents, num_offsprings):
    offsprings = np.empty((num_offsprings, parents.shape[1]))
    crossover_point = int(parents.shape[1]/2)
    crossover_rate = 0.8
    i=0
    while (parents.shape[0] < num_offsprings):
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        x = rd.random()
        if x > crossover_rate:
            continue
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        offsprings[i,0:crossover_point] = parents[parent1_index,0:crossover_point]
        offsprings[i,crossover_point:] = parents[parent2_index,crossover_point:]
        i=+1
    return offsprings

def mutation(offsprings):
    mutants = np.empty((offsprings.shape))
    mutation_rate = 0.4
    for i in range(mutants.shape[0]):
        random_value = rd.random()
        mutants[i,:] = offsprings[i,:]
        if random_value > mutation_rate:
            continue
        int_random_value = randint(0,offsprings.shape[1]-1)
        if mutants[i,int_random_value] == 0 :
            mutants[i,int_random_value] = 1
        else :
            mutants[i,int_random_value] = 0
    return mutants

def optimize(weight, value, population, pop_size, num_generations, threshold):
    parameters, fitness_history = [], []
    num_parents = int(pop_size[0]/2)
    num_offsprings = pop_size[0] - num_parents
    for i in range(num_generations):
        fitness = cal_fitness(weight, value, population, threshold)
        fitness_history.append(fitness)
        parents = selection(fitness, num_parents, population)
        offsprings = crossover(parents, num_offsprings)
        mutants = mutation(offsprings)
        population[0:parents.shape[0], :] = parents
        population[parents.shape[0]:, :] = mutants

    print('\nLast generation: \n{}\n'.format(population))
    fitness_last_gen = cal_fitness(weight, value, population, threshold)
    print('Fitness of the last generation: \n{}\n'.format(fitness_last_gen))
    max_fitness = np.where(fitness_last_gen == np.max(fitness_last_gen))
    parameters.append(population[max_fitness[0][0],:])
    return parameters, fitness_history

parameters, fitness_history = optimize(weight, value, initial_population, pop_size,
num_generations, knapsack_threshold)
print('The optimized parameters for the given inputs are: \n{}'.format(parameters))
selected_items = item_number * parameters
print('\nSelected items that will maximize the knapsack without breaking it:')
for i in range(selected_items.shape[1]):
  if selected_items[0][i] != 0:
     print('{}'.format(selected_items[0][i]))
```

OUTPUT:



```
The list is as follows:
Item No.   Weight   Value
1          6         437
2          4         69
3          3         552
4          5         423
5          10         240
6          9         666
7          9         268
8          5         237
9          12         208
10          1          205

Population size = (8, 10)
Initial population:
[[0 1 1 0 1 1 1 1 0 1]
 [0 0 0 1 1 1 1 1 0 0]
 [1 0 1 0 0 0 0 0 0 1]
 [1 1 0 0 0 0 0 1 1 1]
 [1 1 0 1 0 0 0 0 0 0]
 [1 1 1 1 0 1 0 1 1 0]
 [0 1 1 0 0 0 1 0 0 1]
 [0 1 1 1 0 0 0 1 0 1]]
```



```
Last generation:
[[1 0 1 1 0 1 1 0 0 1]
 [1 0 1 1 0 1 1 0 0 1]
 [1 0 1 1 0 1 1 0 0 1]
 [1 0 1 1 0 1 1 0 0 1]
 [1 0 1 1 0 1 1 0 0 0]
 [1 0 1 1 0 1 1 0 0 1]
 [1 0 1 1 0 0 1 0 0 1]
 [1 0 1 1 0 1 1 0 0 1]]

Fitness of the last generation:
[2551 2551 2551 2551 2346 2551 1885 2551]

The optimized parameters for the given inputs are:
[array([1, 0, 1, 1, 0, 1, 1, 0, 0, 1])]

Selected items that will maximize
the knapsack without breaking it:   1
                                    3
                                    4
                                    6
                                    7
                                    10
```

Learning Outcomes: Students should have the ability to

LO1: Understand the problem Genetic Algorithm

Course Outcomes: Upon completion of the course students will be able to understand problem formulation in AISC using Genetic Algorithm

Conclusion: Hence we have successfully understood the genetic algorithm and implemented the same in solving a real-life problem using its concepts.

Viva Questions:

Ques.1 What do you understand by Genetic Algorithm?

Ques. 2. Explain the basic steps in GA?

Ques. 3. Write types of problem discussed in detail.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				