4/3/2020

GENETIC ALGORITHM

A solution of Knapsack problem

Name of Student: N. I. Md. Ashafuddula
Roll: 18204016
Program: M. Sc. Engineering in CSE Program
Course: CSE-6406
Course Title: Advanced Algorithm Design and Analysis

Introduction:

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.
Genetic Algorithms have the ability to deliver a “good-enough” solution “fast-enough”. This makes genetic algorithms to use in solving optimization problems.

Genetic algorithm follows each of the following steps:

1. Population initializations
2. Fitness function
3. Cross over
4. Mutation
5. Servivor Selction
6. Go to step 2 untill termination condition reached
7. Terminate and return the best outcome

GA algorithm:

```
GA():
  initialize population
  find fitness of population
  while (termination criteria is reached) do
    parent selection
    crossover with probability pc
    mutation with probability pm
    decode and fitness calculation
    survivor selection
    find best
  return best
```

Knapsack problem:

Given,

No. of item, n= 6.

Max weight, m = 20.

W_i = {7,3,8,3,9,11} &

$P_{i\,=}$ {3,5,4,9,10,6}

We need to find max $\sum_{1\le i\le n} Pi * Xi$, subject to $\sum_{1\le i\le n} Wi * Xi \le m$. Where $1\le$ Xi $\le$ n & $1\le$ i $\le$ n.

CODE:

```
import numpy as np

import random

import time

start_time = time.time()

itemize = [[7,3],[3,5],[8,4],[3,9],[9,10],[11,6]] #_W_&_P_

maxW = 20

no_of_Population = len(itemize)

population_No = 8

c,r = no_of_Population,population_No;

population = [[0 for x in range(c)] for y in range(r)] #Mat(8*6)

#print(population)

fitness_col = 2

fitness = [[0 for x in range(fitness_col)] for y in range(r)] #weight & Profit

#print(fitness)

def callRandPos(c):

    mat1 = random.sample(range(0,no_of_Population), c)#position can't be copied means, returns unique position

    return (mat1)


def initPopulation(population,fitness,r,c):

    for i in range(0,r):

        finessWeight = 0

        finessProfit = 0

        ran = int(random.uniform(1,c)) % 4 #no. of 1 in a row ,say = 3

        ran2 = callRandPos(ran+1)

        #print(ran,' ',ran2)

        for j in range(0,len(ran2)):

            finessWeight += itemize[ran2[j]][0]

            finessProfit += itemize[ran2[j]][1]
            population[i][ran2[j]] = 1
        fitness[i][0] = finessWeight
        fitness[i][1] = finessProfit
    #print('Epoch: ',i+1,'>> population: ',population,' Fitness: ',fitness)
    return(population,fitness)
population,fitness = initPopulation(population,fitness,r,c)
for i in range(0,no_of_Population):
    print('Itemize W-P : ',i,' > ',itemize[i])
for i in range(0,population_No):
    print('Population : ',population[i],'fitness : ',fitness[i])
def final_result(population,fitness):
    maxPr = 0 #min
    loc = 0
    for i in range(0,len(population)):
        if(fitness[i][0]<= maxW and maxPr < fitness[i][1] ):
            maxPr = fitness[i][1]
            loc = i
    print('Final result : ',population[loc],' Fitness : ',fitness[loc])
def calcFitness(cross_Result_Population,no_of_Population):
    fitness_Cross = [[0 for x in range(fitness_col)] for y in range(population_No)] #weight & Profit
    #print('Cross result : ',fitness_Cross)
    for i in range(0,len(cross_Result_Population)):
        fitnessWeight = 0
        fitnessProfit = 0
        for j in range(0,no_of_Population):
            if(cross_Result_Population[i][j] == 1):
                fitnessWeight += itemize[j][0]
                fitnessProfit += itemize[j][1]
        fitness_Cross[i][0] = fitnessWeight
        fitness_Cross[i][1] = fitnessProfit
        #print('Fitness cal : ',fitnessWeight,' ',fitnessProfit)
    #print(fitness_Cross)
    return(fitness_Cross)
def tournament_Selection_Call(randSelection,fitness,maxW,selection_no): #[[6, 4, 1], [3, 4, 0], [5, 7, 2], [1, 3, 5]]
    best_Cross = [[0 for x in range(2)] for y in range(len(randSelection))] #r = 4, c = 2
    minP = 10000
    dis = []
    store_i = 0
    for j in range(0,len(randSelection)):
        posDiscard = 0
        for i in range(0,selection_no):
            if(fitness[randSelection[j][i]][0]> maxW or fitness[randSelection[j][i]][1] < minP):
                minP = fitness[randSelection[j][i]][1]
                store_i = i
        posDiscard = randSelection[j][store_i]
        dis.append(posDiscard)

    for i in range(0,len(randSelection)):
        x = 0
        for j in range(0,selection_no):
            if(dis[i] != randSelection[i][j]):
                best_Cross[i][x] = randSelection[i][j]
                x += 1
    #print('Discard : ',dis)
    #print('rand Cross : ',randSelection,' best Cross :',best_Cross)
    return(best_Cross)
def crossOver_Call(best_Cross,population,population_No,no_of_Population,fitness_col,r):
    #best = [[3, 1], [6, 1], [1, 0], [3, 0]]
    rand = int(random.uniform(2,6))
    no_Cross = len(best_Cross) * 2
    cross_Result = [[0 for x in range(no_of_Population)] for y in range(no_Cross)] #8
    #cross_Fitness_Result
    Q = 0
    for j in range(0,len(best_Cross)):
        for i in range(0,no_of_Population):
            if(i <= rand):
                cross_Result[Q][i] = population[best_Cross[j][0]][i]
                cross_Result[Q+1][i] = population[best_Cross[j][1]][i]
            else:
                cross_Result[Q][i] = population[best_Cross[j][0]][i]
                cross_Result[Q+1][i] = population[best_Cross[j][1]][i]
        Q += 2
    fitness_Cross = calcFitness(cross_Result,no_of_Population)
    #print('fitness Cross : ',fitness_Cross)

    return(cross_Result,fitness_Cross)

def mutaion(cross_Result):
    for i in range(0,len(cross_Result)):
        ran_Mutation_Point = int(random.uniform(0,6))
        if(cross_Result[i][ran_Mutation_Point] == 0):
            cross_Result[i][ran_Mutation_Point] = 1
        else:
            cross_Result[i][ran_Mutation_Point] = 0
    mut_Fitness = calcFitness(cross_Result,no_of_Population)
    return(cross_Result,mut_Fitness)

def newPopulation(mutationPopulation,mutaionFitness,population,fitness,maxW,no_of_Population):
    for i in range(0,len(population)):
        if((mutaionFitness[i][0] < maxW or mutaionFitness[i][0] < fitness[i][0]) and mutaionFitness[i][1] >= fitness[i][1]):
            for j in range(0,no_of_Population):
                population[i][j] = mutationPopulation[i][j]
    return(population)
def selection_N_crossover_N_mutation(population,fitness,population_No,no_of_Population,maxW):
    selection_no = 3
    findCrossPoint = int(population_No / 2)
    randSelection = [[0 for x in range(selection_no)] for y in range(findCrossPoint)]
    for j in range(0,findCrossPoint):
        randSelection[j] = random.sample(range(0,population_No), 3) #[0,4,2] random 3 select
    #print(randSelection)
    best_Cross = tournament_Selection_Call(randSelection,fitness,maxW,selection_no) #[3,2] best two for cross over
    cross_Result,cross_Fitness_Result = crossOver_Call(best_Cross,population,population_No,no_of_Population,fitness_col,r)
    #print('Cross Over : ',cross_Result,'  :  ',cross_Fitness_Result)
    mutationPopulation,mutaionFitness = mutaion(cross_Result)
    #print('Mutation : ',mutationPopulation,'  :  Mutation Fitness : ',mutaionFitness)
    new_population = newPopulation(mutationPopulation,mutaionFitness,population,fitness,maxW,no_of_Population)
    return(new_population)



def GA_knapsack(population,fitness,population_No,no_of_Population,maxW):
    new_population = selection_N_crossover_N_mutation(population,fitness,population_No,no_of_Population,maxW)
    new_fitness = calcFitness(new_population,no_of_Population)
    return(new_fitness,new_population)
define_epoch = 8
for i in range(0,define_epoch):
    fitness,population = GA_knapsack(population,fitness,population_No,no_of_Population,maxW)
    print('Epoch > ',i+1,' ',end = ' ')
    final_result(population,fitness)


print('\nGA Result:')
final_result(population,fitness)
print('Actual output : ',[15,24])
print('Used Time',time.time() - start_time,'s')
```

Input & Output:

```
Itemize W-P :  0  >  [7, 3]
Itemize W-P :  1  >  [3, 5]
Itemize W-P :  2  >  [8, 4]
Itemize W-P :  3  >  [3, 9]
Itemize W-P :  4  >  [9, 10]
Itemize W-P :  5  >  [11, 6]
Population :  [0, 0, 1, 0, 1, 0] fitness :  [17, 14]
Population :  [0, 0, 1, 0, 0, 1] fitness :  [19, 10]
Population :  [1, 0, 0, 1, 0, 1] fitness :  [21, 18]
Population :  [1, 0, 0, 1, 0, 0] fitness :  [10, 12]
Population :  [1, 0, 0, 0, 0, 0] fitness :  [7, 3]
Population :  [0, 0, 0, 0, 0, 1] fitness :  [11, 6]
Population :  [0, 0, 1, 1, 1, 1] fitness :  [31, 29]
Population :  [1, 1, 0, 0, 0, 1] fitness :  [21, 14]
Epoch >  1   Final result :  [0, 0, 1, 0, 1, 0]  Fitness :  [17, 14]
Epoch >  2   Final result :  [0, 0, 1, 1, 1, 0]  Fitness :  [20, 23]
Epoch >  3   Final result :  [0, 0, 1, 1, 1, 0]  Fitness :  [20, 23]
Epoch >  4   Final result :  [0, 0, 1, 1, 1, 0]  Fitness :  [20, 23]
Epoch >  5   Final result :  [0, 0, 1, 1, 1, 0]  Fitness :  [20, 23]
Epoch >  6   Final result :  [0, 1, 0, 1, 1, 0]  Fitness :  [15, 24]
Epoch >  7   Final result :  [0, 1, 0, 1, 1, 0]  Fitness :  [15, 24]
Epoch >  8   Final result :  [0, 1, 0, 1, 1, 0]  Fitness :  [15, 24]

GA Result:
Final result :  [0, 1, 0, 1, 1, 0]  Fitness :  [15, 24]
Actual output :  [15, 24]
Used Time 0.02090907096862793 s
```