Fall 2020: CSCI 4588/5588 Programming

Assignment #1B”.

“Name: Tharani Maaneeivaannan ID:2575198

# Solution to Genetic Algorithm

## Program:

import os

import random

import operator

import matplotlib.pyplot as plt

from textwrap import wrap

class GeneticAlgorithm:

    def \_\_init\_\_(self):

        """

        1. Initialize the Population

        """

        self.newPopulationList = []

    """

    This module generates the Chromosome Orientation/structure in Random for the given sequence

    Input: [gene1,gene2,....,gene\_N]

    Output: [(gene1, (X1, Y1)), (gene2, (X2, Y2)), ...., (gene\_N, (Xn, Yn))]

    """

    def chromosomeOrientation(self,sequence):

        #Assigning the first value of binary sequence to (0,0) position initially

        currentPosition=(0,0)

        chromosome=[(sequence[0],currentPosition)]

        #Creating a dictionary to store assigned coordinates

        assignedCoordinates={currentPosition}

        for gene in sequence[1:]:

            allOptions=[]

            validOptions=[]

            # Adding the right direction option

            allOptions.append((currentPosition[0]+1,currentPosition[1]))

            # Adding the left direction option

            allOptions.append((currentPosition[0]-1,currentPosition[1]))

            # Adding the Up direction option

            allOptions.append((currentPosition[0],currentPosition[1]+1))

            # Adding the Down direction option

            allOptions.append((currentPosition[0],currentPosition[1]-1))

            # filtering valid options from allOptions

            for axis in allOptions:

                if axis not in assignedCoordinates:

                    validOptions.append(axis)

            # Select one Valid position in random and assign the gene as it's position and move forward with the orientation

            forwardPosition = random.choice(validOptions)

            chromosome.append((gene, forwardPosition))

            assignedCoordinates.add(forwardPosition)

            currentPosition = forwardPosition

        return chromosome

    """

    Given a chromosome this function calculates and returns the fitness value

    """

    def calculateFitness(self,chromosome):

        chromosome\_dict={}

        fitness=0

        for indexposition,chromosomedata in enumerate(chromosome):

            chromosome\_dict[chromosomedata[1]]=(indexposition,chromosomedata[0])

        for key,\_ in chromosome\_dict.items():

            #get the fitness value if there is a topological neighbour in right

            fitness = fitness + self.individualFitness(key,(key[0]+1,key[1]),chromosome\_dict)

            #get the fitness value if there is a topological neighbour in left

            fitness = fitness + self.individualFitness(key,(key[0]-1,key[1]),chromosome\_dict)

            #get the fitness value if there is a topological neighbour in up

            fitness = fitness + self.individualFitness(key,(key[0],key[1]+1),chromosome\_dict)

            #get the fitness value if there is a topological neighbour in down

            fitness = fitness + self.individualFitness(key,(key[0],key[1]-1),chromosome\_dict)

        return fitness

    """

    This function returns the individual fitness at a selected axis by comparing

    the topological neighbours

    """

    def individualFitness(self,baseAxis,neighbourAxis,chromosome\_dict):

        #if the neighbouraxis is part of the chromosome orientation

        if neighbourAxis in chromosome\_dict:

            baseGene=chromosome\_dict[baseAxis]

            # print(baseGene)

            neighbourGene=chromosome\_dict[neighbourAxis]

            # print(neighbourGene)

            # checking the topological neighbours in ascending order and eliminating the covalent bonded neighbours

            if (baseGene[0]<neighbourGene[0]) and (abs(baseGene[0]-neighbourGene[0])>1):

                if baseGene[1]=='h' and neighbourGene[1]=='h':

                    return 1

        return 0

    def rouletteWheelSelection(self,population):

        max=0

        current=0

        for i in population:

            max+=i[1]

        selection=random.randint(0,max)

        for i in population:

            current+=i[1]

            if current>selection:

                return i[0]

    def crossover(self,selectedChromosomes,crossoverPosition):

        chromosome1 = selectedChromosomes[0]

        chromosome2 = selectedChromosomes[1]

        # print(chromosome1,chromosome2,crossoverPosition)

        partOfChromosome1=chromosome1[:crossoverPosition+1]

        partOfChromosome2=chromosome2[crossoverPosition+1:]

        partOfChromosome1Axes={i[1] for i in partOfChromosome1}

        #Detecting Previous Direction

        prevDir = None

        Ax = [0,0,0]

        Ay = [0,0,0]

        if chromosome1[crossoverPosition][1][1]==chromosome1[crossoverPosition-1][1][1]:

            if (chromosome1[crossoverPosition-1][1][0] - chromosome1[crossoverPosition][1][0]) == 1:

                prevDir = 'RIGHT'

            else:

                prevDir = 'LEFT'

        else:

            if (chromosome1[crossoverPosition-1][1][1] - chromosome1[crossoverPosition][1][1]) == 1:

                prevDir = 'UP'

            else:

                prevDir = 'DOWN'

        if prevDir == 'RIGHT':

            Ax = [-1,0,0]

            Ay = [0,1,-1]

        elif prevDir == 'LEFT':

            Ax = [1,0,0]

            Ay = [0,1,-1]

        elif prevDir == 'UP':

            Ax = [1,-1,0]

            Ay = [0,0,-1]

        elif prevDir == 'DOWN':

            Ax = [1,-1,0]

            Ay = [0,0,1]

        # performing three crossovers based on the direction of the first chromosome

        for itr in range(3):

            crossoverList = partOfChromosome2

            crossover\_Xdir = chromosome1[crossoverPosition][1][0]+Ax[itr] - chromosome2[crossoverPosition+1][1][0]

            crossover\_Ydir = chromosome1[crossoverPosition][1][1]+Ay[itr] - chromosome2[crossoverPosition+1][1][1]

            crossoverList[0] = (crossoverList[0][0],(chromosome1[crossoverPosition][1][0]+Ax[itr],chromosome1[crossoverPosition][1][1]+Ay[itr]))

            for j in range(len(crossoverList)-1):

                crossoverList[j+1] = (chromosome2[crossoverPosition+j+2][0],(chromosome2[crossoverPosition+j+2][1][0]+ crossover\_Xdir ,chromosome2[crossoverPosition+j+2][1][1]+ crossover\_Ydir))

            crossover\_Axes = {a[1] for a in crossoverList}

            if not self.collision(partOfChromosome1Axes,crossover\_Axes):

                return partOfChromosome1 + crossoverList

            elif itr == 2:

                return None

    def collision(self,axes1,axes2):

        for axis in axes2:

            if axis in axes1:

                return True

        return False

    def mutate(self,chromosome):

        # print(chromosome)

        # Selecting the random index by omitting 1st and last positions.

        # This index will be used for mutation

        randomIndexInChromosome = random.randint(1,len(chromosome)-2)

        # print(randomIndexInChromosome)

        selectedGeneAxis = chromosome[randomIndexInChromosome][1]

        # First Part will remain constant

        firstPart = chromosome[:randomIndexInChromosome+1]

        possibleCollisionAxes = {i[1] for i in firstPart}

        #Second part will be rotated and joined

        secondPart = chromosome[randomIndexInChromosome+1:]

        #90 Degree Rotation

        rotated=[(value,(axis[1]+selectedGeneAxis[0]-selectedGeneAxis[1],selectedGeneAxis[0]+selectedGeneAxis[1]-axis[0])) for value,axis in secondPart]

        rotatedAxes={i[1] for i in rotated}

        if (not(self.collision(possibleCollisionAxes,rotatedAxes))):

            # print("90 Rotated")

            return firstPart+rotated

        #180 Degree Rotation

        rotated=[(value,(2\*selectedGeneAxis[0]-axis[0],2\*selectedGeneAxis[1]-axis[1])) for value,axis in secondPart]

        rotatedAxes={i[1] for i in rotated}

        if (not(self.collision(possibleCollisionAxes,rotatedAxes))):

            # print("180 rotated")

            return firstPart+rotated

        #270 Degree Rotation

        rotated=[(value,(selectedGeneAxis[0]+selectedGeneAxis[1]-axis[1],axis[0]+selectedGeneAxis[1]-selectedGeneAxis[0])) for value,axis in secondPart]

        rotatedAxes={i[1] for i in rotated}

        if (not(self.collision(possibleCollisionAxes,rotatedAxes))):

            # print("270 Rotated")

            return firstPart+rotated

        # If all rotations failed

        return None

    def plotFigure(self,chromosome,filename,sequence,bestfitness,maxFitness):

        x = [x[1][0] for x in chromosome]

        y = [y[1][1] for y in chromosome]

        x0 = [x[1][0] for x in chromosome if x[0]=='h' ]

        x1 = [x[1][0] for x in chromosome if x[0]=='p' ]

        y0 = [y[1][1] for y in chromosome if y[0]=='h' ]

        y1 = [y[1][1] for y in chromosome if y[0]=='p' ]

        plt.figure(figsize=(10,8))

        plt.title("\n".join(wrap("Sequence: '%s', Best Fitness: %d, Max\_Fitness: %d"%(sequence,bestfitness,maxFitness))))

        plt.plot(x,y,linewidth=3.0)

        hmarker = dict(color='0',marker='o',markersize=10,linewidth=0,label='h')

        plt.plot(x0,y0,\*\*hmarker)

        pmarker = dict(color='0',marker='o',markersize=10,fillstyle='none',linewidth=0,label='p')

        plt.plot(x1,y1,\*\*pmarker)

        plt.grid(True)

        # plt.show()

        plt.legend(loc="best")

        plt.savefig('figure-%s'%(str(filename)))

        plt.close()

    def GA\_Main(self,proteinList,sIdx):

        sequence=proteinList[0]

        maxFitness=proteinList[1]

        # print(sequence)

        # print(fitness)

        chromosomeList = []

        populationList = []

        elitePopulation = []

        crossoverPopulation = []

        randomPopulation = []

        topFitnessList = []

        generation = 0

        # binarySequence=self.getBinarySequence(sequence)

        while (len(chromosomeList)<300):

            try:

                chromosomeList.append(self.chromosomeOrientation(list(sequence)))

            # When there is situation of collision while forming chromosome randomly, Indexerror occurs

            # I catch the exception and drop that structure

            except IndexError:

                # print("No Valid Options to select")

                continue

        """

        2. Compute Fitness of Population for all Chromosome Ci

        """

        for chromosome in chromosomeList:

            populationList.append((chromosome, self.calculateFitness(chromosome)))

        # print(self.populationList[0],len(self.populationList))

        """

        3. Sort the Population in descending order based on their fitness

        """

        populationListSorted = sorted(populationList,key=operator.itemgetter(1),reverse=True)

        # print(\*populationListSorted,sep='\n')

        #currentGenerationPopulation =

        """

        4. Examine: C1/Progress or Max\_gen, Exit Condition

        """

        while generation < 80000:

            print("Generation Number #%d"%(generation))

            if generation > 0:

                currentGenerationPopulationSorted = sorted(self.newPopulationList,key=operator.itemgetter(1),reverse=True)

                self.newPopulationList = []

            else:

                currentGenerationPopulationSorted = populationListSorted

            topFitness = currentGenerationPopulationSorted[0][1]

            topFitnessList.append(topFitness)

            print("Top\_Fitness: %d"%(topFitness))

            """

            Breaking conditions defined below

            """

            # break if the program reached 80000 generations

            if generation>80000:

                break

            # break the program if the maximum fitness is attained

            if topFitness == maxFitness:

                self.plotFigure(currentGenerationPopulationSorted[0][0],sIdx,sequence,topFitness,maxFitness)

                break

            # break the program if there is no progress in the fitness value after 100 iterations,

            # if the last 30 top fitness is same exit the program

            if len(topFitnessList)>100:

                if len(set(topFitnessList[-30:]))==1:

                    print("No Progress - So Breaking")

                    self.plotFigure(currentGenerationPopulationSorted[0][0],sIdx,sequence,topFitness,maxFitness)

                    break

            """

            5. Taken 5% of Elite and form a New Population

            """

            elitePopulation = currentGenerationPopulationSorted[:15]

            self.newPopulationList = self.newPopulationList + elitePopulation

            # print("After Elite : %d"%(len(self.newPopulationList)))

            """

            6. 80% crossover chromosomes and fill the New Population

            """

            beforeCrossoverPopulation = currentGenerationPopulationSorted[15:255]

            crossoverResultList = []

            while (len(crossoverResultList)<240):

                try:

                    selectedChromosomes=[]

                    for \_ in range(2):

                        selectedChromosomes.append(self.rouletteWheelSelection(beforeCrossoverPopulation))

                    crossoverResult=self.crossover(selectedChromosomes,random.randint(2,len(selectedChromosomes[0])-2))

                    if not (crossoverResult==None):

                        crossoverResultList.append(crossoverResult)

                except:

                    # print(e)

                    continue

            for xover in crossoverResultList:

                crossoverPopulation.append((xover,self.calculateFitness(xover)))

            # print(crossoverPopulation)

            crossoverPopulationSorted = sorted(crossoverPopulation,key=operator.itemgetter(1),reverse=True)

            # print(self.crossoverPopulation)

            self.newPopulationList = self.newPopulationList + crossoverPopulationSorted

            # print("After Crossover : %d"%(len(self.newPopulationList)))

            """

            7. Fillup Pop2 randomly

            """

            randomPopulation = []

            forRandomPopulation = crossoverPopulationSorted + currentGenerationPopulationSorted[255:]

            for \_ in range(30):

                selectedIndex = random.randint(0,len(forRandomPopulation)-1)

                randomPopulation.append(forRandomPopulation[selectedIndex])

            self.newPopulationList = self.newPopulationList + randomPopulation

            # print("After Random : %d"%(len(self.newPopulationList)))

            """

            8. Mutate the 5% of the Non elite chromosome in the

            population1 and fill it in Pop2

            """

            mutationPopulation = []

            while len(mutationPopulation)<15:

                selectedChromosomeIndex = random.randint(15,len(currentGenerationPopulationSorted)-1)

                selectedChromosometoMutate = currentGenerationPopulationSorted[selectedChromosomeIndex]

                # print(selectedPopulationForMutation[selectedChromosomeIndex])

                mutationResult = self.mutate(selectedChromosometoMutate[0])

                    # print(mutationResult)

                if not (mutationResult == None):

                    mutationPopulation.append((mutationResult,self.calculateFitness(mutationResult)))

            self.newPopulationList = self.newPopulationList + mutationPopulation

            # print("After Mutation : %d"%(len(self.newPopulationList)))

            """

            9.Increase Generation and Goto Step2

            """

            # self.newPopulationList = elitePopulation + randomPopulation

            # print(currentGenerationPopulation[:30],sep='\n')

            # print(len(self.newPopulationList))

            elitePopulation = []

            crossoverPopulation = []

            randomPopulation = []

            generation += 1

def main():

    with open("Input.txt") as f:

        content = f.readlines()

    seqValues = []

    fitnessValues = []

    contents = [x.strip() for x in content]

    for line in contents:

        if len(line)>0:

            if "Comment" in line:

                continue

            if "Seq" in line:

                seqValues.append(line.replace("Seq = ",""))

            if "Fitness =" in line:

                fitnessValues.append(abs(int(line.replace("Fitness = ",""))))

    proteinList = list(zip(seqValues,fitnessValues))

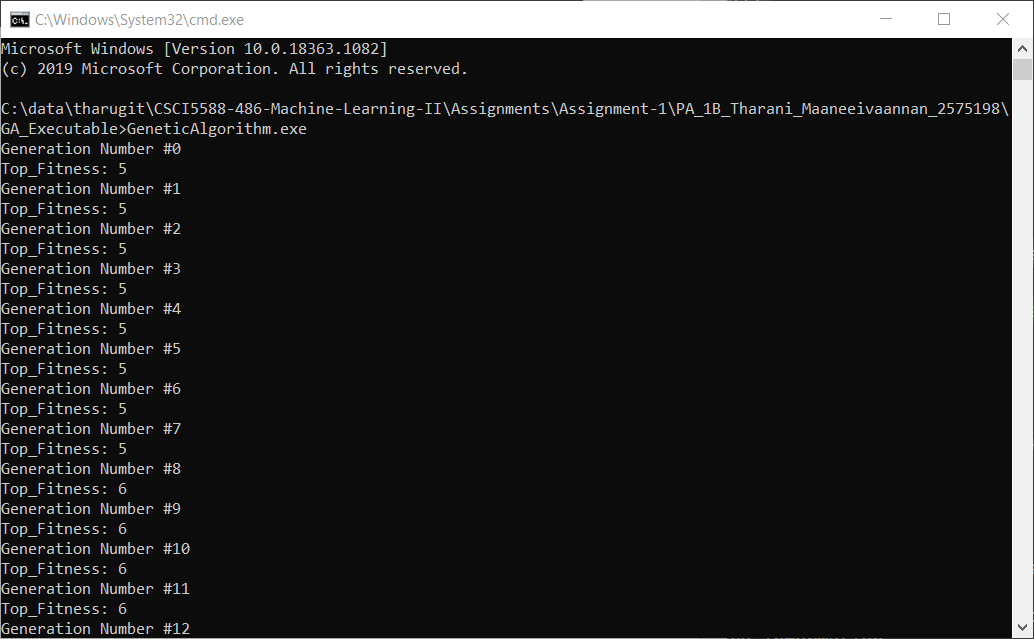
    for idx,value in enumerate(proteinList):

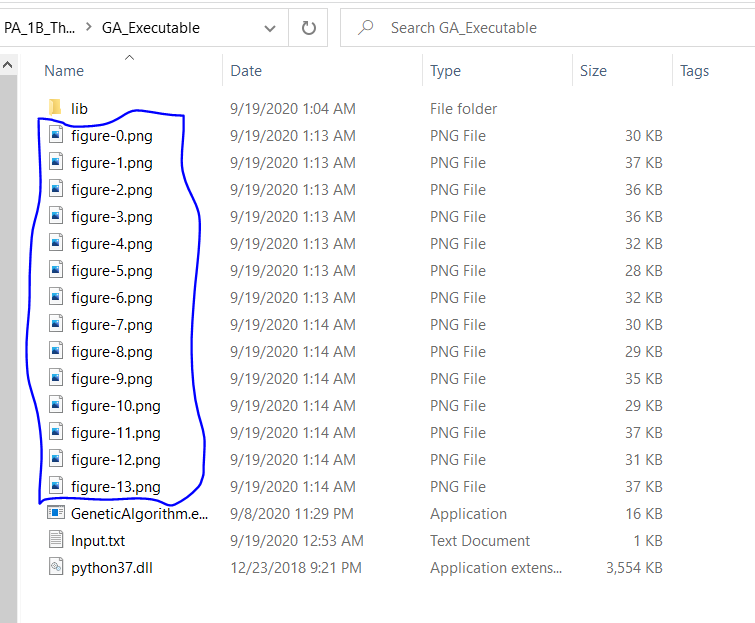
        GeneticAlgorithm().GA\_Main(value,idx)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

## Output:





Protein sequence figures will be generated in the same folder