Fall 2020: CSCI 4588/5588 Programming

Assignment #1”.

“Name: Tharani Maaneeivaannan ID:2575198

# Solution to Part 1 – Hill Climbing Algorithm

## Program:

import numpy as np

import os

import collections

class HillClimbing:

    """

    Initializing the constants

    STRING\_LENGTH and NUMBER\_OF\_ITERATIONS

    """

    def \_\_init\_\_(self):

        self.STRING\_LENGTH = 40

        self.MAX = 100

    """

    The below function creates a random array of 0's and 1's

    given the array size and how many ones you want in that array

    """

    def CreateRandomArray(self,arsize,ones):

        onesArray = np.ones(ones, dtype=np.int)

        zerosArray = np.zeros(arsize-ones, dtype=np.int)

        wholeArray = np.concatenate((onesArray, zerosArray), axis = 0)

        np.random.shuffle(wholeArray)

        return wholeArray

    """

    The below function calculates the fitness value for the given random array

    It counts the number of 1's in the given array and applies that in the function and returns

    the fitness value

    """

    def CalculateFitness(self,onescount):

        return abs(13\*onescount-170)

    def getOnesCount(self,arr):

        return collections.Counter(arr)[1]

    """

    Given an array of length asize this function returns a array list of

    one bit changed neighbours of length asize

    """

    def getNeighbours(self,arr,asize):

        neighbours = []

        for index in range(asize):

            temparr = list(arr)

            temparr[index] = 1 - arr[index]

            neighbours.append(temparr)

        return neighbours

    """

    Given a list of neighbourhood arrays

    this function returns largest fitness value and the Largest neighbour VN

    """

    def GetLargestFV(self,arr):

        largestFV = 0

        for a in arr:

            currentOnesCount = self.getOnesCount(a)

            currentFV = self.CalculateFitness(currentOnesCount)

            if currentFV > largestFV:

                largestFV = currentFV

                largestVN = a

        return largestFV, largestVN

def main():

    hc = HillClimbing()

    #reset the algorithm for MAX times

    t = 0

    while t < hc.MAX:

        """

        Selection of random array with zero's and one's evenly distributed to get the

        Global maximum value and Local Maximum value

        """

        if t%2 == 0:

            randomVC = hc.CreateRandomArray(hc.STRING\_LENGTH,np.random.randint(0,20))

        else:

            randomVC = hc.CreateRandomArray(hc.STRING\_LENGTH,np.random.randint(20,hc.STRING\_LENGTH))

        """

        Calculating the number of ones for the random VC and Evaluating the fitness value

        for VC

        """

        randomOnescount = hc.getOnesCount(randomVC)

        funtionvalueRandomVC = hc.CalculateFitness(randomOnescount)

        local = False

        while(not(local)):

            neighbours = hc.getNeighbours(randomVC,hc.STRING\_LENGTH)

            functionValuelargestVN, largestVN = hc.GetLargestFV(neighbours)

            if funtionvalueRandomVC < functionValuelargestVN:

                funtionvalueRandomVC = functionValuelargestVN

                randomVC = largestVN

            else:

                local = True

        if t < 99:

            print(funtionvalueRandomVC,end='')

            print(',',end='')

        else:

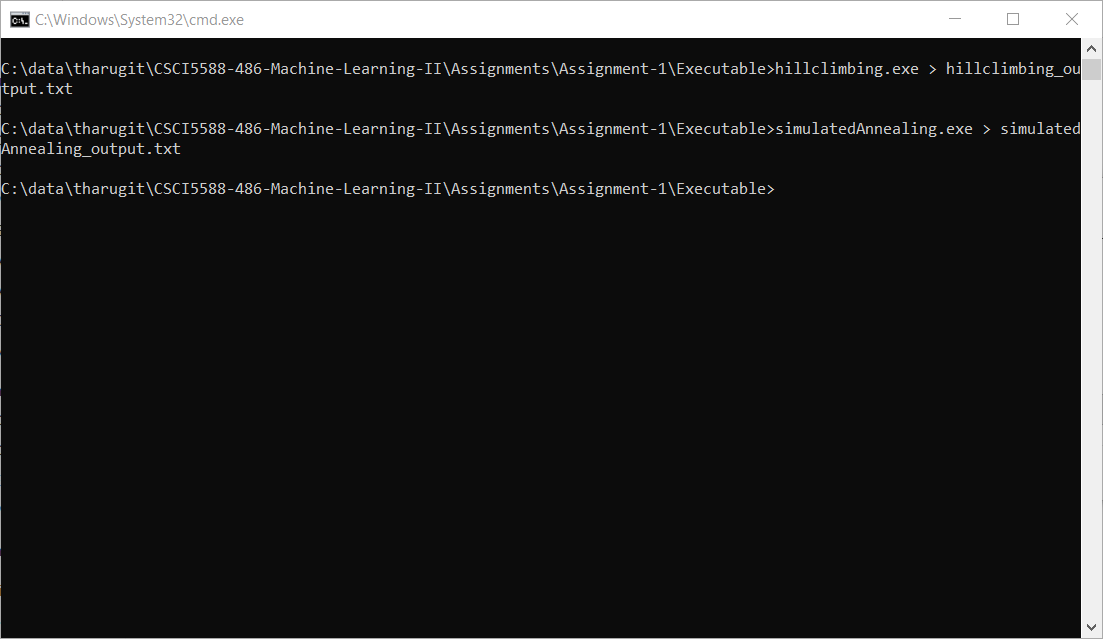
            print(funtionvalueRandomVC)

        t += 1

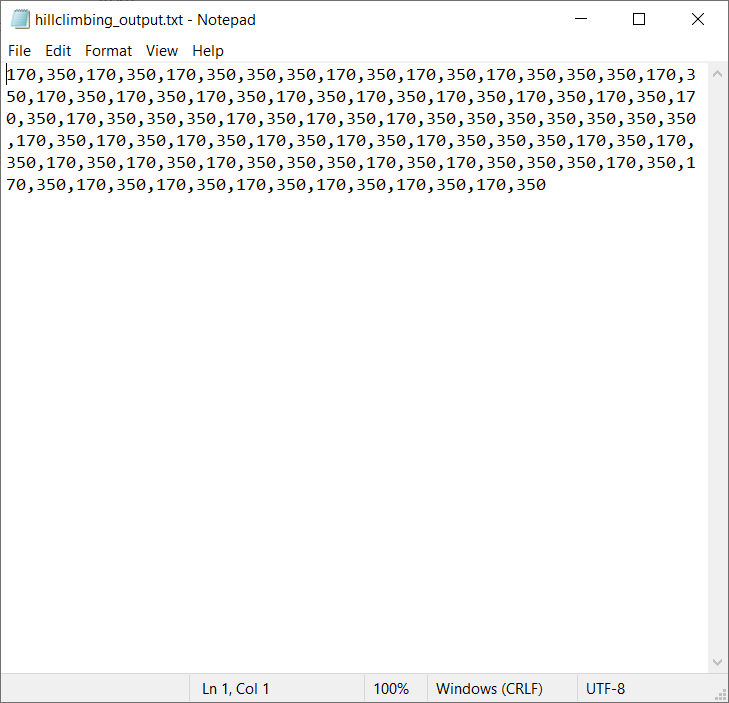
if \_\_name\_\_=="\_\_main\_\_":

    main()

## Output:



Output file is stored inside Executable/Hillclimbing\_output.txt



# Solution for Part 2: Simulated Annealing

## Program:

import numpy as np

import os

import collections

from math import exp

class SimulatedAnnealing:

    """

    Initializing the constants

    STRING\_LENGTH and NUMBER\_OF\_ITERATIONS

    """

    def \_\_init\_\_(self):

        self.STRING\_LENGTH = 50

        self.MAX = 200

    """

    The below function creates a random array of 0's and 1's

    given the array size and how many ones you want in that array

    """

    def CreateRandomArray(self,arsize,ones):

        onesArray = np.ones(ones, dtype=np.int)

        zerosArray = np.zeros(arsize-ones, dtype=np.int)

        wholeArray = np.concatenate((onesArray, zerosArray), axis = 0)

        np.random.shuffle(wholeArray)

        return wholeArray

    """

    The below function calculates the fitness value for the given random array

    It counts the number of 1's in the given array and applies that in the function and returns

    the fitness value

    """

    def CalculateFitness(self,onescount):

        return abs(14\*onescount-190)

    def getOnesCount(self,arr):

        return collections.Counter(arr)[1]

    """

    Given an array of length asize this function returns a array list of

    one bit changed neighbours of length asize

    """

    def getNeighbours(self,arr,asize):

        neighbours = []

        for index in range(asize):

            temparr = list(arr)

            temparr[index] = 1 - arr[index]

            neighbours.append(temparr)

        return neighbours

def main():

    sa = SimulatedAnnealing()

    #reset the algorithm for MAX times

    t = 0

    Temperature = 100

    randomVC = sa.CreateRandomArray(sa.STRING\_LENGTH,np.random.randint(0,sa.STRING\_LENGTH))

    """

    Calculating the number of ones for the random VC and Evaluating the fitness value

    for VC

    """

    randomOnescount = sa.getOnesCount(randomVC)

    functionvalueRandomVC = sa.CalculateFitness(randomOnescount)

    while t < sa.MAX:

        neighbours = sa.getNeighbours(randomVC,sa.STRING\_LENGTH)

        """

        For each neighbour in the neighbours list and we calculating its fitness value and comparing

        it with fitness value of randomVc and the new randomVC is updated based on two different conditions

        """

        for neighbour in neighbours:

            onescount = sa.getOnesCount(neighbour)

            functionValueneigbourVN = sa.CalculateFitness(onescount)

            expcalc = exp((functionValueneigbourVN - functionvalueRandomVC)/Temperature)

            if functionvalueRandomVC < functionValueneigbourVN:

                functionvalueRandomVC = functionValueneigbourVN

                randomVC = neighbour

            elif np.random.uniform(0,1)< expcalc:

                functionvalueRandomVC = functionValueneigbourVN

                randomVC = neighbour

        #Print the output

        if t < (sa.MAX-1):

            print(functionvalueRandomVC,end='')

            print(',',end='')

        else:

            print(functionvalueRandomVC)

        #Decrease the temperature by 5 percent

        Temperature = Temperature \* 0.95

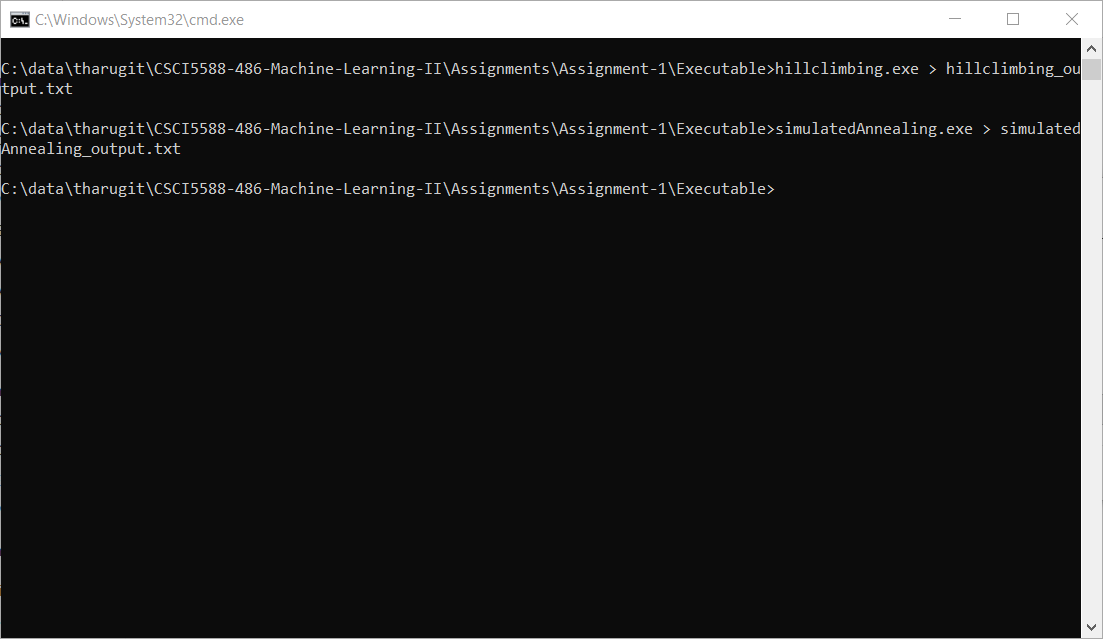
        #Increase the iteration by 1 until MAX

        t += 1

if \_\_name\_\_=="\_\_main\_\_":

    main()

## Output:



Output file is stored inside Executable/simulatedAnnealing\_output.txt. Based on the random value selection. The output value either evolves to Local Maximum or Global Maximum

