Spring 2023: CSCI 4588/5588 Programming Assignment #1

Name: Aviral Kandel
ID: 2630609

1. **Hill Climbing Algorithm**

Code:

```
import random
def neighborhood(binary_string)                          # neighbor function
   list_a = list(binary_string)
   main_flipped = []
   #print(list_a)
   for i in range(len(list_a)):
      if list_a[i] == '0':
         flip_bin = list_a.copy()                          # copy to flip_bin if the bit is 0
         flip_bin[i] = '1'                                 # copy but make that bit to 1
         main_flipped.append(''.join(flip_bin))            # append the rest of the bits

      else:  # same for the 0 bit
         flip_bin = list_a.copy()
         flip_bin[i] = '0'
         main_flipped.append(''.join(flip_bin))
   main_flipped.append(''.join(binary_string))          # added all the neighboring strings into main_flipped list
   return main_flipped

def fitness(binary):
   ones = binary.count('1')
   return abs(13 * ones - 170)

def hill_climbing_algorithm(binary_bit):
   current = binary_bit                          # intitially both current and best bits are the binary bits
   best = binary_bit
   while True:
      #print(current)
      neighbors = neighborhood(current)                 # found neighbors by calling the user defined function
      #print(neighbors)
      #print(neighbors)
      best_neighbor = current
      best_fitness = fitness(best_neighbor)     #fitness function
      for x in neighbors:                           # for loop used to find best function for each neighbors
         neighbor_fitness = fitness(x)
         if neighbor_fitness > best_fitness:
            best_neighbor = x
```

```
            best_fitness = neighbor_fitness
        if best_fitness> fitness(best):                    # used for improving fitness function to either global or local maxima
            current = best_neighbor
            best = best_neighbor
        else:
            return best
maximum = 0
maximum_list = []

for i in range(100):
        binary = "".join(random.choices(["0", "1"], k=40))      # to generate random binary number with 40 bits
    result = hill_climbing_algorithm(binary)
    result_fitness = fitness(result)
    maximum_list.append(result_fitness)
    if result_fitness > maximum:                                # getting the best maximum value
        maximum = result_fitness
        best_result = result

with open("Output.txt", "w") as file:                    # writing the code into output text file
    file.write(", ".join(str(x) for x in maximum_list))
```

Output:

utput.txt

```
350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 170, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350,
350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350,
350, 350, 350, 170, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350, 350,
350, 350, 350, 350, 350, 350, 350
```

Explanation:
Hill-climbing algorithm is an optimization algorithm which compares the fitness function for each neighborhood, and selects the bit with maximum fitness function. The problem with hill climbing is that it might get stuck in the local maxima and declare local maximum as its global maximum. In our code as well, in some of the iterations, we are stuck at 170(local maxima), while in some of them it iterates into 350(global maximum).

2. Simulated Annealing Algorithm

Code:

```
import random
import math


def neighborhood(binary_string):                    # generate neighborhood bits by flipping each bit at a time
    list_a = list(binary_string)
    main_flipped = []
    #print(list_a)
```

```
    for i in range(len(list_a)-1):
        if list_a[i] == '0':
            flip_bin = list_a.copy()                    # copy to flip_bin if the bit is 0
            flip_bin[i] = '1'                           # copy but make that bit to 1
            main_flipped.append(''.join(flip_bin))      # append the rest of the bits

        else:  # same for the 0 bit
            flip_bin = list_a.copy()
            flip_bin[i] = '0'
            main_flipped.append(''.join(flip_bin))
    main_flipped.append(''.join(binary_string))
    return main_flipped

def fitness(binary):                                    # fitness function
    ones = binary.count('1')
    return abs(14 * ones - 190)

def simulated_annealing_algorithm(binary_bit):          # user defined simulated annealing algorithm
    temp = 20                                           # initial temperature
    cooling_rate = 0.01
    current = binary_bit
    best = binary_bit

    while temp > 1:
        neighbors = neighborhood(current)               # all neighbors are found by calling user defined function
        best_neighbor = best
        best_fitness = fitness(best)
        for x in neighbors:                             # for loop used to find best fitness function in each neighbor
            neighbor_fitness = fitness(x)
            if neighbor_fitness > best_fitness:
                best_neighbor = x
                best_fitness = neighbor_fitness
            elif (random.uniform(0,1) < math.exp((neighbor_fitness - best_fitness) / temp)):   # checks again with
#probabilistic approach
                best_neighbor = x
                best_fitness = neighbor_fitness

        temp = temp * (1-cooling_rate)
        if best_fitness > fitness(best):
            current = best_neighbor
            best = best_neighbor
     return best

maximum = 0
maximum_list = []
for i in range(200):
    binary_bit = "".join(random.choices(["0", "1"], k=50))    # generate random binary number with 50 bits
    result = simulated_annealing_algorithm(binary_bit)
    result_fitness = fitness(result)
```

```
    maximum_list.append(result_fitness)

    if result_fitness > maximum:
        maximum = result_fitness
        best_result = result
with open("Output.txt", "w") as file:                        # writing the maximum list in the output text
    file.write(", ".join(str(x) for x in maximum_list))
```

Output:

Output.txt

```
1   510, 496, 510, 510, 510, 496, 496, 510, 510, 510, 510, 510, 510, 496, 496, 510, 496, 510, 510, 510, 510, 496, 510, 496, 510, 510, 496, 510, 496, 496, 510,
    510, 496, 510, 496, 510, 496, 496, 510, 496, 510, 496, 510, 510, 496, 496, 496, 510, 496, 496, 496, 496, 496, 510, 510, 496, 496, 510, 496, 496, 496, 510,
    510, 510, 496, 496, 510, 510, 510, 496, 496, 510, 496, 496, 496, 496, 496, 510, 496, 510, 496, 510, 496, 496, 510, 496, 510, 496, 510, 510, 496, 510, 496,
    496, 496, 510, 496, 496, 510, 496, 510, 496, 510, 510, 510, 510, 510, 510, 496, 496, 496, 510, 510, 510, 496, 496, 510, 496, 510, 496, 496, 496, 496, 496,
    510, 496, 496, 496, 510, 510, 496, 496, 496, 510, 496, 496, 510, 496, 510, 510, 496, 496, 510, 510, 496, 510, 496, 510, 510, 510, 496, 496, 496, 510, 496,
    496, 510, 496, 496, 496, 510, 510, 510, 510, 496, 510, 496, 510, 510, 510, 510, 496, 496, 510, 496, 496, 510, 496, 496, 496, 510, 496, 496, 496, 510, 510,
    496, 510, 496, 496, 510, 496, 510, 496, 510, 510, 510, 496, 510, 496
```

Explanation:

Simulated Annealing algorithm is an optimization algorithm which compares the fitness function for each neighborhood. Simulated annealing gets its name from the process of slowly cooling metal, applying this idea to the data domain. The problem of hill climbing is solved by simulated annealing algorithm. In this algorithm, even the fitness function of one of the neighbor is less than the other, it doesn't reject that bit. But instead, it again checks by comparing a random number with exponential of((neighbor_fitness – best_fitness) / temperature), where temperature is slowly cooled in the next iteration. Our code doesn't get stuck in the local maxima, instead finds the global maxima (510). The global maxima occur