PS4

Graded

Student

Chetan Hiremath

Total Points

100 / 100 pts

Question 1

(no title) **10** / 10 pts

✔ **+ 10 pts** Correct

Question 2

(no title) **10** / 10 pts

✔ **+ 10 pts** Correct

Question 3

(no title) **20** / 20 pts

✔ **+ 20 pts** Correct

Question 4

(no title) **30** / 30 pts

✔ **+ 30 pts** Correct

Question 5

(no title) **30** / 30 pts

✔ **+ 30 pts** Correct

Question assigned to the following page: 1

1a. There are many factors that are considered for evolving agents to control an elevator. These factors are the average waiting time for a passenger before the elevator's arrival, the average waiting time for a passenger before the elevator's destination to the passenger's chosen floor, and the maximum waiting times. There is a possibility that these times are infinite because some passengers aren't serviced during early stages of evolving an elevator controller. Factors like number of unserviced passengers, energy efficiency, smooth passenger ride, equal load distribution, and elevator active time are included to specify this fitness function.

b. The factors of evolving agents that control stop lights on a city main street are average and maximal waiting times at stop lights for through traffic and average and maximal waiting times at stop lights for side-street traffic. The fitness function needs to consider the tradeoff if through traffic and side-street traffic are facing any conflicts. Factors like traffic volume/flow, pedestrian safety, emergency vehicle priority, fuel efficiency, and prevention of any traffic conflicts are included to specify this fitness function.

Question assigned to the following page: 2

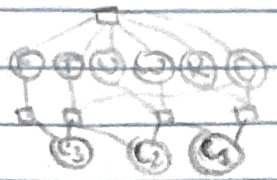
2. X-Variables- Classes, $X = \{X_1, \ldots, X_n\}$.
D/Domains- Time Slots of classes. $D = D_j = \{D_1, \ldots, D_3\}$
C/Constraints- 1 class is taught by exactly one professor. 1 class is assigned to one classroom. 1 class is assigned to a time slot.

Question assigned to the following page: 2

No professor can exactly teach 2 classes at the same time slot. No classes are assigned to the same classroom too.
C = {(professor, class) | one professor p teaches one class c}

Question assigned to the following page: 3

3. TWO
+TWO
FOUR



$O+O = R+10C_1$
~~$C_1+W+W = U+10C_2$~~
$C_2+T+T = O+10C_3$
$C_3 = F$

F=1 (F doesn't have a leading 0, so 1 is chosen.)

F=1
O=4 (O must be an even number because T+T=2T<5, so 4 is chosen.)

F=1
O=4
R=8 (O+O=2O=2(4)=8=R, so 8 is chosen.)

F=1
O=4
R=8
T=7 (T+T=2T=2(7)=14, the first digit is used as a carry 1 for F, which is 1. Second digit 4 is [letter] O, so 7 is chosen.)

F=1
O=4
R=8
T=7
U=6 (U must be an even number because 4+W=2W<9, so 6 is chosen.)

F=1
O=4
R=8
T=7
U=6
W=3 (W+W=2W=6, so W=3. 3 is chosen.)

```
  TWO     7 3 4
+ TWO  →  + 7 3 4
 FOUR    1 4 6 8
```

min-conflicts make sense for this type of problem since they are used to solve constraint satisfaction problems/local search problems. You choose a variable and min-conflict values randomly and repeat.

Question assigned to the following page: 4

4.

Sources- This code is modified, used, and borrowed from EECS 649: Lec 7's Local Search's Evolutionary Algorithms and GA Pseudocode.

```
import numpy as np
import random

#Objective function
def F(x):
    return 4 + (2 * x) + (2 * np.sin(20 * x)) - (4 * x**2)

#Mutation operation on the interval [0,1]
def mutate(x, epsilon=0.01):
    mutation_type = np.random.choice(['subtract', 'copy', 'add'], p=[0.3,
0.4, 0.3])
    if mutation_type == 'subtract':
        return max(0, x - epsilon)
    elif mutation_type == 'copy':
        return x
    else:
        return min(1, x + epsilon)

#Crossover operation (Convex combination of 2 individuals for 0 <= a <= 1)
def crossover(x, y, a):
    return (a * x) + ((1 - a) * y)

#Fitness-proportional selection (roulette selection)
def roulette_selection(population, fitness_values):
    probabilities = fitness_values / np.sum(fitness_values)
    selected_index = np.random.choice(len(population), p=probabilities)
    return population[selected_index]
```

Question assigned to the following page: 4

```
#Main optimization loop
def optimization(N, epsilon, generations):
    population = np.linspace(0, 1, N)
    for generation in range(generations):
        fitness_values = F(population)
        new_population = []
        for _ in range(N):
            parent1 = roulette_selection(population, fitness_values)
            parent2 = roulette_selection(population, fitness_values)
            a = np.random.rand()  # Crossover parameter
            child = crossover(parent1, parent2, a)
            child = mutate(child, epsilon)
            new_population.append(child)
        population = np.array(new_population)
    best_solution = population[np.argmax(F(population))]
    return best_solution, F(best_solution)

#Code Parameters
N = 10
epsilon = 0.01
generations = [i*0.01 for i in range(1,100)]

#Run optimization to get final results.
best_solution, best_fitness = optimization(N, epsilon, len(generations))
print("Best x:", best_solution)
print("Best F(" + str(best_solution) + "):", best_fitness)
```

Question assigned to the following page: 4

Result-

```
Best x: 0.39081939846232305
Best F(0.39081939846232305): 6.169266470878865
```

I have used the parameters that are shown in the Python Code and run the code, and the results show me that the best x and the best F(x) are 0.39081939846232305 and 6.169266470878865 respectively. This result means that the maximum value F(x) is approximately 6.169 when x =~ 0.39, so the one-dimensional optimization experiment is successful with the interval [0,1] and N=10 individuals. Then, I have included the crossover in the Python Code. It finds the maximum value F(x) at x approximately too. If I run the Python code to generate results, then the results will be random due to the random library. However, the results will find maximum values at random values of x.

Question assigned to the following page: 5

5a.
Sources- This code is modified, used, and borrowed from 8queens.pynb's Python code and Aimacode Algorithms' Python Code Repository.

RRHC General Implementation's Python Code-

```
import numpy as np
import random

#Return the number of queens conflicting in grid.
def fitness(queens, N):
    c = 0
    for q in range(len(queens)):
        for q_y in range(len(queens)):
            q_x = queens[q_y]
            if q_y != q: # Check if queens are not attacking each other.
                if q_x == queens[q]:
                    c += 1
                if abs(q - q_y) == abs(queens[q] - q_x):
                    c += 1
    return (N*(N-1))/2 - c

#Code Parameters
N = 8 #Grid size or number of queens
i = 0 #Evaluation count
```

Question assigned to the following page: 5

```
#Use RRHC algorithm for the n-queens problem to print the string of digits.
for x in range(1):
    count = 0
    success = False
    while not success:
        queens = []
        count +=1
        i+=1
        #Random start state for queen n
        queens = []
        for n in range(N):
            queens.append(random.randint(0,N-1))
        #If there is a solution, then it breaks the loop and prints the
solution.
        if fitness(queens, N) == N*(N-1)/2:
            print(queens)
            success = True
print('Total number of fitness evaluations: ' + str(i))
print('Average:', (i/100))
```

Question assigned to the following page: 5

Result-

```
[0, 6, 3, 5, 7, 1, 4, 2]
Total number of fitness evaluations: 353644
Average: 3536.44
```

b. I have tested several routines by this implementation, and I am getting the appropriate solutions randomly since I am using the random library.

c. I have gathered statistics by running it for 100 times and calculating the average of the number of fitness evaluations after the solution is found. The results show me that the string of digits is accurate because the queens won't attack each other if I use these digits that represent the positions on the board. If I run the Python code to generate results for the solutions, then the results will be random due to the random library. The results will manage to find the solutions, their respective fitness evaluations' counts, and their respective fitness evaluations' counts' averages.