**CSE422\_03\_Lab\_Assignment02\_Turnitin\_Summer2024**

**Name : Md. Anwar Hossain | Student ID : 22101710**

import random

input\_file = 'input.txt'

output\_file = 'output.txt'

candidate\_count = 10

max\_generations = 100

variation\_probability = 0.01

def generate\_population(course\_nums, timeslot\_nums, candidate\_count):

population = []

len\_chromosome = course\_nums \* timeslot\_nums

for \_ in range(candidate\_count):

chromosome = ''

for i in range(len\_chromosome):

chromosome += random.choice('01')

population.append(chromosome)

return population

def fitness\_calculation(chromosome, course\_nums, timeslot\_nums):

overlap\_penalty = 0

consistency\_penalty = 0

class\_schedule = []

for i in range(timeslot\_nums):

timeslot = chromosome[i\*course\_nums: (i+1)\*course\_nums]

class\_schedule.append([int(bit) for bit in timeslot])

# print(class\_schedule)

for timeslot in class\_schedule:

course\_schedule\_nums = sum(timeslot)

if course\_schedule\_nums > 1:

overlap\_penalty += (course\_schedule\_nums - 1)

# print(overlap\_penalty)

for x in range(course\_nums):

time\_schedule\_nums = sum(timeslot[x] for timeslot in class\_schedule)

if time\_schedule\_nums != 1:

consistency\_penalty += abs(time\_schedule\_nums - 1)

# print(consistency\_penalty)

total\_penalty = overlap\_penalty + consistency\_penalty

fitness = -total\_penalty

# print(fitness)

return fitness

def parents\_selection(population, all\_fitness):

lowest\_fitness = min(all\_fitness)

if lowest\_fitness < 0:

all\_fitness = [fitness - lowest\_fitness for fitness in all\_fitness]

# print(all\_fitness)

if sum(all\_fitness) == 0:

return random.sample(population, 2)

parents = random.choices(population, weights=all\_fitness, k=2)

# print(parents)

return parents

def crossover\_method(p1, p2):

crossing\_point = random.randint(1, len(p1)-1)

c1 = p1[:crossing\_point] + p2[crossing\_point:]

c2 = p2[:crossing\_point] + p1[crossing\_point:]

return c1, c2

# two point crossover methods for part 2

def two\_point\_crossover(p1, p2):

point1 = random.randint(1, len(p1) - 2)

point2 = random.randint(point1 + 1, len(p1) - 1)

c1 = p1[:point1] + p2[point1:point2] + p1[point2:]

c2 = p2[:point1] + p1[point1:point2] + p2[point2:]

return c1, c2

def mutation\_function(chromosome, variation\_probability):

# print(chromosome)

mutated\_chromosome = ''

for bit in chromosome:

if random.random() > variation\_probability:

mutated\_chromosome += bit

else:

if bit == '0':

mutated\_chromosome += '1'

else:

mutated\_chromosome += '0'

return mutated\_chromosome

def Genetic\_Algorithm(course\_nums, timeslot\_nums, candidate\_count, max\_generations, variation\_probability):

population = generate\_population(course\_nums, timeslot\_nums, candidate\_count)

best\_chromosome = None

best\_fitness = float('-inf')

for \_ in range(max\_generations):

all\_fitness = []

for chromosome in population:

all\_fitness.append(fitness\_calculation(chromosome, course\_nums, timeslot\_nums))

# print(all\_fitness)

if max(all\_fitness) > best\_fitness:

best\_fitness = max(all\_fitness)

best\_chromosome = population[all\_fitness.index(best\_fitness)]

diverse\_population = []

for \_ in range(candidate\_count // 2):

p1, p2 = parents\_selection(population, all\_fitness)

c1, c2 = crossover\_method(p1,p2)

# c1, c2 = two\_point\_crossover(p1, p2) # this one for part 2

# print(c1,c2)

diverse\_population.extend([mutation\_function(c1,variation\_probability), mutation\_function(c2,variation\_probability)])

population = diverse\_population

return best\_chromosome, best\_fitness

def input\_reading(input\_file):

with open(input\_file, 'r') as input:

rows = input.readlines()

course\_nums, timeslot\_nums = map(int, rows[0].strip().split())

course\_names = [row.strip() for row in rows[1:]]

# print(course\_names)

return course\_nums, timeslot\_nums, course\_names

def output\_writing(output\_file, best\_chromosome, best\_fitness, course\_names, timeslot\_nums):

with open(output\_file, 'w') as output:

output.write(f"Best Schedule: {best\_chromosome}\n")

output.write(f"Fitness Value: {best\_fitness}\n")

course\_nums, timeslot\_nums, course\_names = input\_reading(input\_file)

best\_schedule, best\_fitness = Genetic\_Algorithm(course\_nums, timeslot\_nums, candidate\_count, max\_generations, variation\_probability)

output\_writing(output\_file, best\_schedule, best\_fitness, course\_names, timeslot\_nums)