Henry Fundenberger

CS461 Program 2

Dr Jesse Lowe

4/14/2023

**Code Section**

from scipy.special import softmax

import numpy as np

import random

import time

mutationRate = 100

facilitators = ["Lock", "Glen", "Banks", "Richards", "Shaw", "Singer", "Uther", "Tyler", "Numen", "Zeldin"]

# Define classes and their details

classes = {

    "SLA100A": {"expectedEnrollment": 50, "preferredFacilitators": ["Glen", "Lock", "Banks", "Zeldin"], "otherFacilitators": ["Numen", "Richards"]},

    "SLA100B": {"expectedEnrollment": 50, "preferredFacilitators": ["Glen", "Lock", "Banks", "Zeldin"], "otherFacilitators": ["Numen", "Richards"]},

    "SLA191A": {"expectedEnrollment": 50, "preferredFacilitators": ["Glen", "Lock", "Banks", "Zeldin"], "otherFacilitators": ["Numen", "Richards"]},

    "SLA191B": {"expectedEnrollment": 50, "preferredFacilitators": ["Glen", "Lock", "Banks", "Zeldin"], "otherFacilitators": ["Numen", "Richards"]},

    "SLA201": {"expectedEnrollment": 50, "preferredFacilitators": ["Glen", "Banks", "Zeldin", "Shaw"], "otherFacilitators": ["Numen", "Richards", "Singer"]},

    "SLA291": {"expectedEnrollment": 50, "preferredFacilitators": ["Lock", "Banks", "Zeldin", "Singer"], "otherFacilitators": ["Numen", "Richards", "Shaw", "Tyler"]},

    "SLA303": {"expectedEnrollment": 60, "preferredFacilitators": ["Glen", "Zeldin", "Banks"], "otherFacilitators": ["Numen", "Singer", "Shaw"]},

    "SLA304": {"expectedEnrollment": 25, "preferredFacilitators": ["Glen", "Banks", "Tyler"], "otherFacilitators": ["Numen", "Singer", "Shaw", "Richards", "Uther", "Zeldin"]},

    "SLA394": {"expectedEnrollment": 20, "preferredFacilitators": ["Tyler", "Singer"], "otherFacilitators": ["Richards", "Zeldin"]},

    "SLA449": {"expectedEnrollment": 60, "preferredFacilitators": ["Tyler", "Singer", "Shaw"], "otherFacilitators": ["Zeldin", "Uther"]},

    "SLA451": {"expectedEnrollment": 100, "preferredFacilitators": ["Tyler", "Singer", "Shaw"], "otherFacilitators": ["Zeldin", "Uther", "Richards", "Banks"]}

}

# Define room capacities

roomCapacities = {

    "Slater 003": 45,

    "Roman 216": 30,

    "Loft 206": 75,

    "Roman 201": 50,

    "Loft 310": 108,

    "Beach 201": 60,

    "Beach 301": 75,

    "Logos 325": 450,

    "Frank 119": 60

}

roomList = list(roomCapacities.keys())

timeSlots = ['10 AM', '11 AM', '12 PM', '1 PM', '2 PM', '3 PM']

# A class to represent an activity

# And hold spcific information about the activity

class Activity:

    def \_\_init\_\_(self, name, expected, preferredFacilitators, otherFacilitators):

        self.name = name

        self.expectedEnrollment = expected

        self.preferredFacilitators = preferredFacilitators

        self.otherFacilitators = otherFacilitators

        self.room = None

        self.time = None

        self.facilitator = None

        self.fitness = 0

    def \_\_str\_\_(self):

        return self.name + " " + self.room + " " + self.time + " " + self.facilitator + " " + str(self.fitness)

# A class to represent a schedule

# And hold spcific information about the schedule

class Schedule:

    def \_\_init\_\_(self, activities = []):

        self.activities = activities

        self.fitness = 0

        self.probability = 0

    def \_\_str\_\_(self):

        return str(self.fitness)

    def printActivities(self):

        for activity in self.activities:

            print(activity)

    # override the less than operator

    def \_\_lt\_\_(self, other):

        return self.fitness < other.fitness

    # Override equals operator

    def \_\_eq\_\_(self, other):

        self.activities == other.activities

        self.fitness == other.fitness

# Random Schedule Generator

def createSchedules(n):

    schedules = []

    for i in range(n):

        schedule = []

        for name, details in classes.items():

            activity = Activity(name, details["expectedEnrollment"], details["preferredFacilitators"], details["otherFacilitators"])

            activity.room = random.choice(list(roomCapacities.keys()))

            activity.time = random.choice(timeSlots)

            activity.facilitator = random.choice(facilitators)

            schedule.append(activity)

        schedules.append(schedule)

    return schedules

# Takes in a singular schedule and calculates the fitness for it

def calcFitnes(schedule):

    # Sechedule is an indivudal shcedule object

    # We can access the activities in the schedule by using schedule.activities

    fitness = 0

    # Check if the room is available at the time

    for activity in schedule.activities:

        for activity2 in schedule.activities:

            if activity.name != activity2.name:

                if activity.room == activity2.room and activity.time == activity2.time:

                    fitness -= 0.5

    # Check if the room is big enough for the class

    for activity in schedule.activities:

            if roomCapacities[activity.room] < activity.expectedEnrollment:

                fitness -= 0.5

            elif roomCapacities[activity.room] > activity.expectedEnrollment \* 3:

                fitness -= 0.2

            elif roomCapacities[activity.room] > activity.expectedEnrollment \* 6:

                print("Room is too big for " + activity.name + " " + activity.room)

                fitness -= 0.4

            else:

                fitness += 0.3

    # Check if the facilitator is a preferred facilitator or other or not

    for activity in schedule.activities:

            if activity.facilitator in activity.preferredFacilitators:

                fitness += 0.5

            elif activity.facilitator in activity.otherFacilitators:

                fitness += 0.2

            else:

                fitness -= 0.1

    # Check if the facilitator is expected to be in many places at one time

    for activity in schedule.activities:

            # Check if the faciliator is expected to be in many places at one time

            facilitatorCount = 0

            for activity2 in schedule.activities:

                if activity.facilitator == activity2.facilitator and activity.time == activity2.time:

                    facilitatorCount += 1

            # If the faciliator only has one class at that time, add 0.2 to the fitness

            if facilitatorCount == 1:

                fitness += 0.2

            # If the faciliator has two or more classes at that time, subtract 0.2 from the fitness

            elif facilitatorCount >= 2:

                fitness -= 0.2

            # Check if the faciliator is overseeing too many activities

            facilitatorCount = 0

            for activity2 in schedule.activities:

                if activity.facilitator == activity2.facilitator:

                    facilitatorCount += 1

            if facilitatorCount > 4:

                fitness -= 0.5

            elif facilitatorCount <= 2 and activity.facilitator != "Tyler":

                fitness -= 0.2

    # If Both Sections SLA100A and SLA100B are scheduled more than 4 hours apart add 0.5 to the fitness

    for activity in schedule.activities:

            if activity.name == "SLA100A":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA100B":

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) > 4:

                            fitness += 0.5

    for activity in schedule.activities:

            if activity.name == "SLA100A":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA100B":

                        if activity.time == activity2.time:

                            fitness -= 0.5

    for activity in schedule.activities:

            if activity.name == "SLA191A":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA191B":

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) > 4:

                            fitness += 0.5

    for activity in schedule.activities:

            if activity.name == "SLA191A":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA191B":

                        if activity.time == activity2.time:

                            fitness -= 0.5

    for activity in schedule.activities:

            if activity.name == "SLA191A" or activity.name == "SLA191B":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA100A" or activity2.name == "SLA100B":

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) == 1:

                            if "Roman" in activity.room and "Beach" in activity2.room:

                                fitness -= 0.4

                            elif "Beach" in activity.room and "Roman" in activity2.room:

                                fitness -= 0.4

                            else:

                                fitness += 0.5

    for activity in schedule.activities:

            if "Roman" in activity.room:

                for activity2 in schedule.activities:

                    if "Beach" in activity2.room and activity.facilitator == activity2.facilitator:

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) == 1:

                            fitness -= 0.4

            elif "Beach" in activity.room:

                for activity2 in schedule.activities:

                    if "Roman" in activity2.room and activity.facilitator == activity2.facilitator:

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) == 1:

                            fitness -= 0.4

    for activity in schedule.activities:

            if activity.name == "SLA191A" or activity.name == "SLA191B":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA100A" or activity2.name == "SLA100B":

                        if abs(timeSlots.index(activity.time) - timeSlots.index(activity2.time)) == 2:

                            fitness += 0.25

    for activity in schedule.activities:

            if activity.name == "SLA191A" or activity.name == "SLA191B":

                for activity2 in schedule.activities:

                    if activity2.name == "SLA100A" or activity2.name == "SLA100B":

                        if activity.time == activity2.time:

                            fitness -= 0.25

    return fitness

# Mutate function takes in a singular schedule and mutates a random part of a single activity

def mutate(schedule):

    if random.randint(0, mutationRate) == 1:

        activity = random.choice(schedule.activities)

        if random.randint(0, 2) == 0:

            activity.time = random.choice(timeSlots)

        elif random.randint(0, 2) == 1:

            activity.room = random.choice(roomList)

        else:

            activity.facilitator = random.choice(facilitators)

    return schedule

# Crossover function takes in two schedules and creates two new schedules by swapping random parts of the schedules

def crossover(schedule1, schedule2):

    # Choose a random index

    breakPoint = random.randint(0, len(schedule1.activities) - 1)

    # Create two new schedules

    parent1 = schedule1.activities[:]

    parent2 = schedule2.activities[:]

    child1 = Schedule(parent1[:breakPoint] + parent2[breakPoint:][:])

    child2 = Schedule(parent2[:breakPoint] + parent1[breakPoint:][:])

    return child1, child2

if \_\_name\_\_ == "\_\_main\_\_":

     # Generates initial 500 schedules

    initialActivities = createSchedules(500)

    initialSchedules = []

    for activity in initialActivities:

        s = Schedule(activity)

        initialSchedules.append(s)

    # Calculate the fitness of each schedule

    # Sum of all the activities fitness

    for schedule in initialSchedules:

        schedule.fitness = calcFitnes(schedule)

    # Keep the best half of the schedules

    initialSchedules.sort(key=lambda x: x.fitness, reverse=True)

    initialSchedules = initialSchedules[:len(initialSchedules)//2]

    i = 0

    # We are going to have a genetic algorithm that will try to find the best schedule

    # The genectic algorithm will have a population of n schedules ( which will be half of the total number of schedules created in the beginning, i.e if we make 500 schedules at the start we pass 250 schedules to the genetic algorithm)

    # The genetic alogrithm will make children out of two random parents, then will have a 1% chance of mutation (i.e. a random activity will be changed, either the time, room or facilitator (use the facilitator list not the preferred facilitator list or the other facilitator list)))

    # We will do this at least 100 times

    print("Starting Genetic Algorithm")

    while True:

        i += 1

        if i % 10 == 0:

            print("Generation: " + str(i))

        if i % 3 == 0:

            mutationRate \*= 2

        schedulesToGoThrough = initialSchedules[:]

        # Make children

        children = []

        usedParents = []

        for j in range(len(initialSchedules)//2):

            # Choose two random parents

            parent1 = random.choice(schedulesToGoThrough[:])

            schedulesToGoThrough.remove(parent1)

            parent2 = random.choice(schedulesToGoThrough[:])

            schedulesToGoThrough.remove(parent2)

            # Make children out of the parents

            child1, child2 = crossover(parent1, parent2)

            # Mutate the children / parents

            child1 = mutate(child1)

            child2 = mutate(child2)

            parent1 = mutate(parent1)

            parent2 = mutate(parent2)

            # Add the children to the list of children

            children.append(child1)

            children.append(child2)

            usedParents.append(parent1)

            usedParents.append(parent2)

        # Calculate the fitness of the children

        for schedule in children:

            schedule.fitness = calcFitnes(schedule)

        # Calculate the fitness of the parents

        for schedule in initialSchedules:

            schedule.fitness = calcFitnes(schedule)

        # Add the children to the list of schedules

        initialSchedules += children

        # Calculate the fitness of the children

        # Sort the schedules by fitness

        initialSchedules.sort(key=lambda x: x.fitness, reverse=True)

        # Print the best fitness

        # Remove the best half of the schedules

        initialSchedules = initialSchedules[:len(initialSchedules)//2]

        # After 100 generations, save the best schedule fitness

        # We do this by saving the probability distribution of the fitness of the schedules

        if i == 100:

            j = 0

            softMaxList = []

            for schedule in initialSchedules:

                softMaxList.append(schedule.fitness)

            probDistro = softmax(softMaxList)

            G100 = probDistro

        # Every iteration after 100, save the probability distribution of the fitness of the schedules

        # And see if the current best schedule is within 1% of the best schedule from gen 100

        if i >= 100:

            j = 0

            softMaxList = []

            for schedule in initialSchedules:

                softMaxList.append(schedule.fitness)

            probDistro = softmax(softMaxList)

            GN = probDistro

            # If current best schedule is within 1% of the best schedule from gen 100, stop the genetic algorithm

            if abs(sum(GN) - sum(G100)) / len(G100) < 0.01:

                break

    # Output the best schedule to a file organized by time

    print("Genetic Algorithm Finished")

    with open('bestSchedule.txt', 'w') as f:

        # write the overall best schedule fitness to the file

        f.write(f"Schedule Fitness: {initialSchedules[0].fitness} \n")

        # write the activities in the schedule to the file in the same order as timeSlots

        for timeSlot in timeSlots:

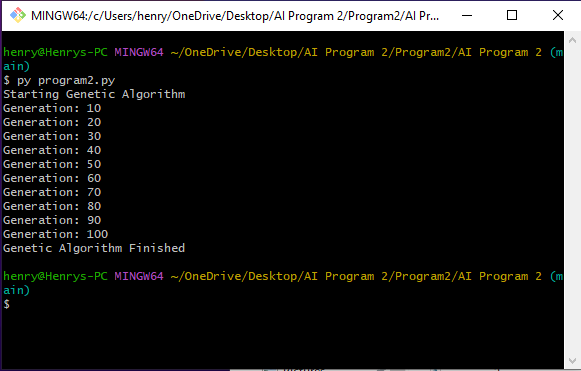
            for activity in initialSchedules[0].activities:

                if activity.time == timeSlot:

                    f.write(f"{activity.name} - {activity.room} - {activity.facilitator} - {activity.time} \n")

**Example Outputs**

**Screenshot of terminal**



Example 1

Text

Description automatically generated

Example 2

Text

Description automatically generated

Example 3

Text

Description automatically generated

**Report Section**

Personally, I would say the first part of this assignment went smoothly. I was able to create however many schedules that were as random as python’s random library could make them. The issues started to rise once I got to the crossbreeding section of the code. At first, I wasn’t quite sure the best way to cross breed. If I should pick randomly or pick from the upper ten or so. I wanted to generate the best offspring, but I also wanted to avoid the possibility of inbreeding. Also, mutations were messing my program up a lot. This was because my mutation rate was way too high, and I didn’t realize that till late in development. Ultimately, I ended up decaying my mutation rate exponentially over time, so mutations were far more likely to happen early on since it seems most mutations are lethal and provided little to no positive change.

I am personally very happy with the schedules that my program is producing. Based off the requirements in the initial document and my fitness function most if not all requirements seem to be getting met when I generate a schedule. So far, I have yet to generate a schedule where one professor is needed at one time slot, a room is too small or far too big, facilitators rarely travel from beach to roman right after the previous class. As it seems no two classes are in the same room at the same time. I think overall the important aspects of a schedule such as a professor being at multiple places at one time, or room double booking, which would make a schedule impossible, have yet to occur and my program does a consistent job at making sure schedules like that are penalized and are not generated. I believe there are always exceptions, and it is entirely possible that a poor schedule could be generated if say a mutation happens very late in life despite it being unlikely.

When it comes to improving this program, I think it works great for this problem on a small scale, but if we were to turn this loose on a bigger problem with a much bigger data sample, it would take far too long. Some ways to improve time could be how we keep the best parents and children between generations, maybe instead of keeping the best upper half we have the upper half compete amongst themselves in a tournament style of staying alive, where only those who “live” can stay in the gene pool. Another option would be parallelization. The fitness function is the largest section and is made up of the most nested for loops out of any section in my code. We could speed this process up quite a bit if parts ran in parallel. If we utilized multi-threading to run say three different parts at once that could be beneficial. We have one section calculating the fitness of professors, one section calculating the fitness of time, and one section calculating the fitness of the room. Having these three sections run at the same time and then converge to calculate one fitness score could produce an answer much quicker than waiting for the one thread to get through the whole section and do everything sequentially.

There is also one aspect of fitness that I think should be added. Dr. Tyler works on the school board, my schedule doesn’t penalize him for taking more than 2 classes at the moment, but if he is on the board I feel as if it should because if he is assigned the acceptable max number of classes, that being four, he would have a lot of time taken out of his day that I think could conflict with his other responsibilities. So, adding a penalization for having more than two or three classes a day for Dr. Tyler I think could be an interesting check to add.

The last thing I would like to add about this project is that overall, it was a very fun project to learn how to implement. I see how this could be turned loose on a lot of other problems that have constraints. I was even thinking about trying to figure out if this could be turned loose on the Einstein problem, however I think that would almost require too much mutation for this program, as every aspect of the word bank is needed. But overall, it has been my favorite program and by far has been one of the more interesting parts of this AI course for me.