***Practical 6***

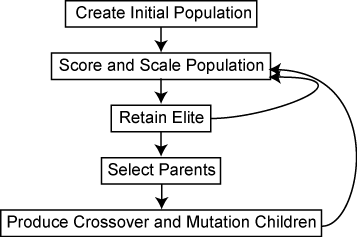
**Aim:** *Implement Genetic Algorithms for Staff Planning.*

***Theory:***

**What Is the Genetic Algorithm?**

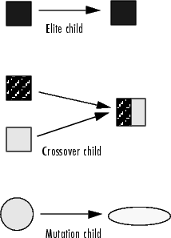
The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of *mixed integer programming*, where some components are restricted to be integer-valued.

This flow chart outlines the main algorithmic steps. For details, see How the Genetic Algorithm Works.



The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

* *Selection rules* select the individuals, called *parents*, that contribute to the population at the next generation. The selection is generally stochastic, and can depend on the individuals' scores.
* *Crossover rules* combine two parents to form children for the next generation.
* *Mutation rules* apply random changes to individual parents to form children.



***Code:***

import numpy as np

import pandas as pd

staff\_planning = [

    [[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0, 10]],

    [[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0, 10]],

    [[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0, 10]],

    [[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0, 10]],

    [[0, 0, 10],[1, 0, 10],[2, 0, 10],[3, 0, 10],[4, 0, 10],[5, 0, 10],[6, 0, 10],[7, 0, 10],[8, 0, 10],[9, 0, 10],[10, 0, 10]]

]

hourlystaff\_needed = np.array([

    [0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

    [0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

    [0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

    [0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2],

    [0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 6, 6, 2, 2, 2, 6, 6, 6, 2, 2, 2, 2]

])

"""

Employee Present: analyse whether the employee is present yes or no on a given time

Based on the employee list of 3 (id, start time, duration)

"""

def employee\_present(employee, time):

    employee\_start\_time = employee[1]

    employee\_duration = employee[2]

    employee\_end\_time = employee\_start\_time + employee\_duration

    if (time >= employee\_start\_time) and (time < employee\_end\_time):

        return True

    return False

"""

convert a staff planning to a staff-needed plannig

The employee planning is organised per employee, the staff-needed planning is the number of employees working per hour

The staff-needed planning is based on the employee planning and will allow to calculate the difference with the staff-needed

It doesnt work overnight, but our shop isnt open at night anyway

"""

def staffplanning\_to\_hourlyplanning(staff\_planning):

    hourlystaff\_week = []

    for day in staff\_planning:

        hourlystaff\_day = []

        for employee in day:

            employee\_present\_hour = []

            for time in range(0, 24):

                employee\_present\_hour.append(employee\_present(employee, time))

            hourlystaff\_day.append(employee\_present\_hour)

        hourlystaff\_week.append(hourlystaff\_day)

    hourlystaff\_week = np.array(hourlystaff\_week).sum(axis = 1)

    return hourlystaff\_week

"""

the cost is calculated as hours understaffed + hours overstaffed

"""

def cost(hourlystaff, hourlystaff\_needed):

    errors = hourlystaff - hourlystaff\_needed

    overstaff = abs(errors[errors > 0].sum())

    understaff = abs(errors[errors < 0].sum())

    overstaff\_cost = 1

    understaff\_cost = 1

    cost = overstaff\_cost \* overstaff + understaff\_cost \* understaff

    return cost

"""

generate an entirely random staff planning for a certain number of days

start time is random between 0 and 23; duration is random between 0 and 10

"""

def generate\_random\_staff\_planning(n\_days, n\_staff):

    period\_planning = []

    for day in range(n\_days):

        day\_planning = []

        for employee\_id in range(n\_staff):

            start\_time = np.random.randint(0, 23)

            duration = np.random.randint(0, 10)

            employee = [employee\_id, start\_time, duration]

            day\_planning.append(employee)

        period\_planning.append(day\_planning)

    return period\_planning

# An example of the code until here

# show the random initialization of the week planning

random\_staff\_planning = generate\_random\_staff\_planning(n\_days = 5, n\_staff = 11)

random\_staff\_planning

# show the cost of this random week planning

cost(staffplanning\_to\_hourlyplanning(random\_staff\_planning), hourlystaff\_needed)

"""

create a parent generation of n parent plannings

"""

def create\_parent\_generation(n\_parents, n\_days = 7, n\_staff = 11):

    parents = []

    for i in range(n\_parents):

        parent = generate\_random\_staff\_planning(n\_days = n\_days, n\_staff = n\_staff)

        parents.append(parent)

    return parents

"""

for each iteration, select randomly two parents and make a random combination of those two parents

by applying a randomly generated yes/no mask to the two selected parents

"""

def random\_combine(parents, n\_offspring):

    n\_parents = len(parents)

    n\_periods = len(parents[0])

    n\_employees = len(parents[0][0])

    offspring = []

    for i in range(n\_offspring):

        random\_dad = parents[np.random.randint(low = 0, high = n\_parents - 1)]

        random\_mom = parents[np.random.randint(low = 0, high = n\_parents - 1)]

        dad\_mask = np.random.randint(0, 2, size = np.array(random\_dad).shape)

        mom\_mask = np.logical\_not(dad\_mask)

        child = np.add(np.multiply(random\_dad, dad\_mask), np.multiply(random\_mom, mom\_mask))

        offspring.append(child)

    return offspring

def mutate\_parent(parent, n\_mutations):

    size1 = parent.shape[0]

    size2 = parent.shape[1]

    for i in range(n\_mutations):

        rand1 = np.random.randint(0, size1)

        rand2 = np.random.randint(0, size2)

        rand3 = np.random.randint(1, 2)

        parent[rand1,rand2,rand3] = np.random.randint(0, 10)

    return parent

def mutate\_gen(parent\_gen, n\_mutations):

    mutated\_parent\_gen = []

    for parent in parent\_gen:

        mutated\_parent\_gen.append(mutate\_parent(parent, n\_mutations))

    return mutated\_parent\_gen

def is\_acceptable(parent):

    return np.logical\_not((np.array(parent)[:,:,2:] > 10).any()) #work more than 10 hours is not ok

def select\_acceptable(parent\_gen):

    parent\_gen = [parent for parent in parent\_gen if is\_acceptable(parent)]

    return parent\_gen

def select\_best(parent\_gen, hourlystaff\_needed, n\_best):

    costs = []

    for idx, parent\_staff\_planning in enumerate(parent\_gen):

        parent\_hourly\_planning = staffplanning\_to\_hourlyplanning(parent\_staff\_planning)

        parent\_cost = cost(parent\_hourly\_planning, hourlystaff\_needed)

        costs.append([idx, parent\_cost])

    print('generations best is: {}, generations worst is: {}'.format(pd.DataFrame(costs)[1].min(), pd.DataFrame(costs)[1].max()))

    costs\_tmp = pd.DataFrame(costs).sort\_values(by = 1, ascending = True).reset\_index(drop=True)

    selected\_parents\_idx = list(costs\_tmp.iloc[:n\_best,0])

    selected\_parents = [parent for idx, parent in enumerate(parent\_gen) if idx in selected\_parents\_idx]

    return selected\_parents

"""

the overall function

"""

def gen\_algo(hourlystaff\_needed, n\_iterations):

    generation\_size = 500

    parent\_gen = create\_parent\_generation(n\_parents = generation\_size, n\_days = 5, n\_staff = 11)

    for it in range(n\_iterations):

        parent\_gen = select\_acceptable(parent\_gen)

        parent\_gen = select\_best(parent\_gen, hourlystaff\_needed, n\_best = 100)

        parent\_gen = random\_combine(parent\_gen, n\_offspring = generation\_size)

        parent\_gen = mutate\_gen(parent\_gen, n\_mutations = 1)

    best\_child = select\_best(parent\_gen, hourlystaff\_needed, n\_best = 1)

    return best\_child

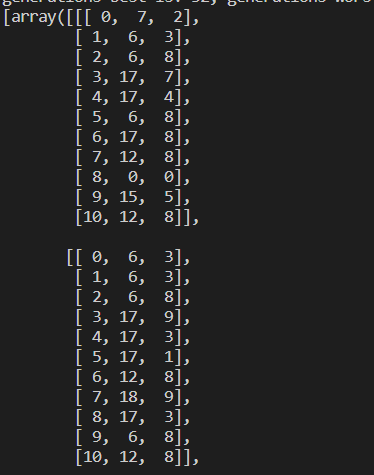
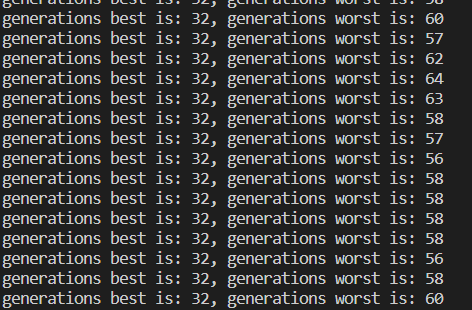
best\_planning = gen\_algo(hourlystaff\_needed, n\_iterations = 100)

print(best\_planning)

print(staffplanning\_to\_hourlyplanning(best\_planning[0]))

print(hourlystaff\_needed)

***Output:***

******