#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Feb 21 00:33:42 2024

@author: hango

"""

from pyomo.environ import \*

import numpy as np

import pandas as pd

import math

#Import file

file\_name ='Group 37-IB9190 - Data.xlsx’

Class=pd.read\_excel(file\_name, 'Class',index\_col=0)

Timetable = pd.read\_excel(file\_name, 'Timetable',index\_col=0)

Room = pd.read\_excel(file\_name, 'Room',index\_col=0)

# Create input

student = Class.index

classes = Class.columns

social\_distancing\_rate = 0.33

room\_capacity = Room['Capacity']\*social\_distancing\_rate

M = 2

time = Timetable.index

w\_TE=1

w\_TD = 0.25

w\_SSE = 0

E = 221\*social\_distancing\_rate

#Create model

model = ConcreteModel()

# Create variables

model.x = Var(student,range(M),domain=Binary)

model.s = Var(time,range(M),domain=NonNegativeIntegers)

model.e = Var(range(M),classes, domain=NonNegativeIntegers)

model.g = Var(range(M),classes, domain=NonNegativeReals)

model.ss = Var(domain=NonNegativeIntegers)

# Objective function

model.Obj = Objective(expr=w\_TE\*sum(model.e[j,k] for j in range(M) for k in classes)+w\_TD\*sum(model.g[j,k] for j in range(M) for k in classes)+w\_SSE\*model.ss,sense=minimize)

# Constraints

## Constraint 1: **Unique Group Assignment**

def constr1\_rule(model,i):

return sum(model.x[i,j] for j in range(M)) == 1

model.Constr1 = Constraint(student,rule=constr1\_rule)

## Constraint 2: **Excess Class Capacity**

def constr2\_rule(model,k,j):

return sum(model.x[i,j]\*Class.loc[i,k] for i in student) - room\_capacity[k] <= model.e[j,k]

model.Constr2 = Constraint(classes,range(M),rule=constr2\_rule)

## Constraint 3: **Allocation Equality**

def constr31\_rule(model,k,j):

return -model.g[j,k]<=sum(model.x[i,j]\*Class.loc[i,k] for i in student) - sum(Class.loc[i,k] for i in student)/M

model.Constr31 = Constraint(classes,range(M),rule=constr31\_rule)

def constr32\_rule(model,k,j):

return sum(model.x[i,j]\*Class.loc[i,k] for i in student) - sum(Class.loc[i,k] for i in student)/M <= model.g[j,k]

model.Constr32 = Constraint(classes,range(M),rule=constr32\_rule)

## Constraint 4: **Simultaneous Excess**

def constr4\_rule(model,t,j):

return model.s[t,j] - E <= model.ss

model.Constr4 = Constraint(time,range(M),rule=constr4\_rule)

## Constraint 5: **Surplus Simultaneous Excess**

def constr5\_rule(model,t,j):

return sum(Timetable.loc[t,k]\*model.e[j,k] for k in classes) <= model.s[t,j]

model.Constr5 = Constraint(time,range(M), rule=constr5\_rule)

# Solve problem

solver=SolverFactory('gurobi')

solver.options['MIPGap'] =0.00

results= solver.solve(model, tee=True)

# Print the objective value

optimal\_sol = round(model.Obj(),2)

print(optimal\_sol)

# Print the group allocation

group\_allocation = pd.DataFrame()

for i in student:

for j in range(M):

group\_allocation.loc[i,j] = model.x[i,j]()

# Print TE

TE =0

for j in range(M):

for k in classes:

TE+=model.e[j,k]()

# Print TD

TD=0

for k in classes:

for j in range(M):

TD += model.g[j,k]()

# Print SSE

SSE = model.ss()

print('TE:',round(TE,2))

print('TD:', round(TD,2))

print('SSE:',round(SSE,2))