project-prototype

March 14, 2022

Authors: Kazmer Nagy-Betegh, Clara Moreno Sanchez, Jasmine Zhang, Sophia Kalusche, Yingjin He, Abdullah Rehman

1 AM13 Group Project

[ ]:

```
import gurobipy as gp
from gurobipy import GRB,quicksum, Model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import os
import sys
import csv
import math
import ipywidgets as widgets
import ipydatetime
from IPython.display import display
import datetime
import panel as pn
```

1.1 Loading the data

[ ]:

```
# read electives data
courses_instances = pd.read_excel('data/CourseSessionsList MAM2022.xlsx')
courses_instances.head()

# clean rooms
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('Virtual␣
 ↪Online', '')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace(' ', '')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('SOC', '')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('SusxPlc',␣
 ↪'')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace(',', '')
```

```python
courses_instances[courses_instances["Session Type"] == "Lecture"].Rooms.unique()

courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('PB-LAB', 'PBLab')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('NBLT12', 'LT12')
courses_instances['Rooms'] = courses_instances['Rooms'].str.replace('WLT', 'LT12')
courses_instances[courses_instances["Session Type"] == "Lecture"].Rooms.unique()

# read capacity data
capacity = pd.read_excel('data/capacity.xlsx')
capacity.drop(['Unnamed: 0'], axis=1, inplace=True)
rooms = capacity.LT.to_list()
rooms[0] = 'LT1'
courses_instances_lecture = courses_instances[(courses_instances["Session Type"] == "Lecture")]

# drop rows with rooms not in capacity
courses_instances_lecture = courses_instances_lecture[courses_instances_lecture["Rooms"].isin(rooms)]

courses_instances_lecture.Rooms.unique()
# create schedule table for each day
rooms = capacity.LT.to_list()
rooms[0] = 'LT1'
scheduling_table = pd.DataFrame(columns=['day', 'time']+rooms)

days = courses_instances["Session Date"].unique()

days_24h = []
# create 15 min increment for each day
for d in days:
    day_times = pd.date_range(start=d+str(" 00:00"), end=d+str(" 23:59"), freq='15min')
    days_24h = days_24h + day_times.to_list()
scheduling_table['day'] = days_24h
scheduling_table['time'] = scheduling_table['day'].apply(lambda x: x.strftime('%H:%M'))
scheduling_table.day = scheduling_table.day.apply(lambda x: x.strftime('%Y-%m-%d'))
scheduling_table.sort_values(by=['day', "time"], inplace=True)

# replace NaN with 0
scheduling_table.fillna(0, inplace=True)
scheduling_table.head()
# mark column if it is blocked

for i in range(courses_instances_lecture.shape[0]):
    date = courses_instances_lecture.iloc[i][ "Session Date"]
    start_time = courses_instances_lecture.iloc[i][ "Start Time"]
    end_time = courses_instances_lecture.iloc[i][ "End Time"]
    room = courses_instances_lecture.iloc[i][ "Rooms"]

    scheduling_table.loc[(scheduling_table.day == date) & (scheduling_table.
 ↪time >= start_time) & (scheduling_table.time <= end_time), room] = 1
# find all lt1 equal to 1

scheduling_table.loc[scheduling_table.LT1 == 1]
```

[ ]:

```
              day   time  LT1  LT2  LT3  LT4  LT5  LT6  LT7  LT9  …  LT15  \
417    2021-09-13  08:15    1    0    0    0    0    0    0    0  …     0
418    2021-09-13  08:30    1    0    0    0    0    0    0    0  …     0
419    2021-09-13  08:45    1    0    0    0    0    0    0    0  …     0
420    2021-09-13  09:00    1    0    0    0    0    0    0    0  …     0
421    2021-09-13  09:15    1    0    0    0    0    0    0    0  …     0
…             …    …   …   …  …  …  …  …  …  …  …   …
16378  2022-06-08  14:30    1    0    0    0    0    0    1    0  …     1
16379  2022-06-08  14:45    1    0    0    0    0    0    1    0  …     1
16380  2022-06-08  15:00    1    0    0    0    0    0    1    0  …     1
16381  2022-06-08  15:15    1    0    0    0    0    0    1    0  …     1
16382  2022-06-08  15:30    1    0    0    0    0    0    1    0  …     1

       LT16  LT17  LT18  LT19  PLG01  PBLab  Trans  WLT  RG06
417       0     0     0     0      0      0      0    0     0
418       0     0     0     0      0      0      0    0     0
419       0     0     0     0      0      0      0    0     0
420       0     0     0     0      0      0      0    0     0
421       0     0     0     0      0      0      0    0     0
…      …    …    …    …     …     …     …  …    …
16378     0     1     0     0      0      0      0    0     0
16379     0     1     0     0      0      0      0    0     0
16380     0     1     0     0      0      0      0    0     0
16381     0     1     0     0      0      0      0    0     0
16382     0     1     0     0      0      0      0    0     0

[1116 rows x 23 columns]
```

1.2 Inputs

Please use the widgets to input your desired bookings.

```
[ ]: # room capacity
     capacity_slider = widgets.IntSlider(min=0, max=120, step=1, value=10,␣
      ↪description='Room Capacity:')

     # date picker
     date_picker = widgets.DatePicker(description='Date:', value=datetime.date.
      ↪today())

     # time picker
     default_time = max(pd.Series(datetime.datetime.now()).dt.round('15min').
      ↪map(lambda x: x.strftime("%H:%M")).to_list()[0], '08:00')
     time_picker = widgets.Dropdown(options = pd.date_range(start='7:00', end='22:
      ↪00', freq='15min').map(lambda x: x.strftime("%H:%M")), description='Start␣
      ↪Time:', value =default_time )

     # event duration
     duration_slider = widgets.Dropdown(options = {"1:00":1, "1:30":1.5, "2:00":2,␣
      ↪"2:30":2.5, "3:00":3, "3:30":3.5, "4:00":4, "4:30":4.5, "5:00":5},␣
      ↪description='Duration (Hours):', style = {'description_width': 'initial'})

     record_entry = widgets.Button(description='Record', button_style='',␣
      ↪icon='check')
     record_out = widgets.Output()


     booking_requests = pd.DataFrame(columns=["time_of_request",'date',␣
      ↪'start_time', 'room_capacity', 'duration'], index=range(0,1))
     # booking_requests = pd.DataFrame()

     # print on button click
     @record_entry.on_click
     def record_entry_click(b):

         with record_out:
             print('Room Capacity:', capacity_slider.value)
             print('Date:', date_picker.value)
             print('Start Time:', time_picker.value)
             print('Duration:', duration_slider.value)
             # save to booking_requests, append row
             booking_requests.loc[booking_requests.index.max()+1] = ([datetime.date.
      ↪today(),date_picker.value, time_picker.value, capacity_slider.value,␣
      ↪duration_slider.value])


             # update table
             print("\n")
```

```
        print("Current Booking Requests:\n")
        print(booking_requests)


print("Use the Below Widgets to input booking requests\n")

display(capacity_slider, date_picker,time_picker,duration_slider,record_entry,␣
 ↪record_out)
```

```
Use the Below Widgets to input booking requests


IntSlider(value=10, description='Room Capacity:', max=120)

DatePicker(value=datetime.date(2022, 3, 14), description='Date:')

Dropdown(description='Start Time:', index=4, options=('07:00', '07:15', '07:30',␣
 ↪'07:45', '08:00', '08:15', '0…

Dropdown(description='Duration (Hours):', options={'1:00': 1, '1:30': 1.5, '2:
 ↪00': 2, '2:30': 2.5, '3:00': 3, …

Button(description='Record', icon='check', style=ButtonStyle())

Output()
```

Please run all upcoming files until further instructions.

[ ]:

```
booking_requests
```

[ ]:

```
  time_of_request        date start_time room_capacity duration
0             NaN         NaN        NaN           NaN      NaN
1      2022-03-14  2022-03-15      09:15            46        1
2      2022-03-14  2022-03-15      12:00            46      2.5
3      2022-03-14  2022-03-14      13:15            46      2.5
4      2022-03-14  2022-03-14      13:15            46      1.5
5      2022-03-14  2022-03-14      13:00            46      1.5
6      2022-03-14  2022-03-14      08:30            46      4.5
7      2022-03-14  2022-03-14      10:30            46      4.5
8      2022-03-14  2022-03-14      15:15            46      4.5
```

[ ]:

```
booking_requests_rf = booking_requests.copy()
booking_requests_rf.drop(booking_requests.index[0], inplace=True)
```

[ ]:

```
booking_requests_rf['LT1'   ] = [ 1 if i <= 100 else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT2'   ] = [ 1 if i <= 45  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT3'   ] = [ 1 if i <= 55  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
```

```
booking_requests_rf['LT4'   ] = [ 1 if i <= 55  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT5'   ] = [ 1 if i <= 47  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT6'   ] = [ 1 if i <= 120 else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT7'   ] = [ 1 if i <= 93  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT9'   ] = [ 1 if i <= 80  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT10'  ] = [ 1 if i <= 81  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT12'  ] = [ 1 if i <= 80  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT14'  ] = [ 1 if i <= 87  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT15'  ] = [ 1 if i <= 86  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT16'  ] = [ 1 if i <= 89  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT17'  ] = [ 1 if i <= 87  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT18'  ] = [ 1 if i <= 100 else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['LT19'  ] = [ 1 if i <= 100 else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['PLG01' ] = [ 1 if i <= 112 else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['PBLab' ] = [ 1 if i <= 91  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['Trans' ] = [ 1 if i <= 59  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['WLT'   ] = [ 1 if i <= 82  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
booking_requests_rf['RG06'  ] = [ 1 if i <= 41  else 0 for i in␣
 ↪booking_requests_rf["room_capacity"]]
```

```
[ ]: booking_requests_rf.date = booking_requests_rf.date.map(lambda x: x.
      ↪strftime("%Y-%m-%d"))
     booking_requests_rf
```

```
[ ]:   time_of_request        date start_time room_capacity duration  LT1  LT2  \
     1      2022-03-14  2022-03-15      09:15            46        1    1    0
     2      2022-03-14  2022-03-15      12:00            46      2.5    1    0
     3      2022-03-14  2022-03-14      13:15            46      2.5    1    0
     4      2022-03-14  2022-03-14      13:15            46      1.5    1    0
```

```
5      2022-03-14  2022-03-14      13:00            46      1.5    1    0
6      2022-03-14  2022-03-14      08:30            46      4.5    1    0
7      2022-03-14  2022-03-14      10:30            46      4.5    1    0
8      2022-03-14  2022-03-14      15:15            46      4.5    1    0

   LT3  LT4  LT5  …  LT15  LT16  LT17  LT18  LT19  PLG01  PBLab  Trans  WLT  \
1    1    1    1  …     1     1     1     1     1      1      1      1    1
2    1    1    1  …     1     1     1     1     1      1      1      1    1
3    1    1    1  …     1     1     1     1     1      1      1      1    1
4    1    1    1  …     1     1     1     1     1      1      1      1    1
5    1    1    1  …     1     1     1     1     1      1      1      1    1
6    1    1    1  …     1     1     1     1     1      1      1      1    1
7    1    1    1  …     1     1     1     1     1      1      1      1    1
8    1    1    1  …     1     1     1     1     1      1      1      1    1

   RG06
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0

[8 rows x 26 columns]
```

1.3 Create Model

Nothing to do here just run the cells. Observe model object for constraint logic.

[ ]:

```
# create model instance for each day optimisation
class BookingOptimiser():

    # create model instance within class
    def __init__(self, date_optimised,scheduling_table, booking_requests):
        self.date_optimised = date_optimised
        self.model = gp.Model(date_optimised)
        self.rooms = scheduling_table.columns[2:]
        self.time = scheduling_table[scheduling_table.day == self.
 ↪date_optimised].time
        self.daily_table = scheduling_table[scheduling_table.day == self.
 ↪date_optimised]

        self.booking_requests = booking_requests[booking_requests.date == self.
 ↪date_optimised]
```

```
        self.room_capacity = {
            'LT1'  : 100,
            'LT2'  : 45 ,
            'LT3'  : 55 ,
            'LT4'  : 55 ,
            'LT5'  : 47 ,
            'LT6'  : 120,
            'LT7'  : 93 ,
            'LT9'  : 80 ,
            'LT10' : 81 ,
            'LT12' : 80 ,
            'LT14' : 87 ,
            'LT15' : 86 ,
            'LT16' : 89 ,
            'LT17' : 87 ,
            'LT18' : 100,
            'LT19' : 100,
            'PLG01': 112,
            'PBLab': 91 ,
            'Trans': 59  ,
            'WLT'  : 82  ,
            'RG06' : 41
        }


        # self.booking_intervals = [self.time_intervals(self.booking_requests.
↪loc[i, "start_time"], self.booking_requests.loc[i, "duration"]) for i in␣
↪self.booking_requests.index]
        # print(self.booking_intervals)

        # create binary booking variables
        self.booking_variables = self.model.
↪addVars(scheduling_table[scheduling_table.day == self.date_optimised].time,␣
↪scheduling_table.columns[2:], vtype=GRB.BINARY, name='booking_variables')

        # create integer of available capacity
        self.available_capacity = self.model.
↪addVars(scheduling_table[scheduling_table.day == self.date_optimised].time,␣
↪lb = 0, vtype=GRB.INTEGER, name='available_capacity')

        # booking request feasibility per room

        self.booking_req = self.model.addVars(self.booking_requests.index, self.
↪rooms, vtype=GRB.BINARY, name='booking_req')
```

```
    def create_constraints(self):
        # create constraints
        # capacity in each hour

        # already booked rooms
        self.model.addConstrs((self.booking_variables[t, r] >= self.daily_table.
↪loc[self.daily_table.time == t, r] for t in self.time for r in self.rooms),␣
↪name='already_booked_rooms')

        # capacity each time interval
        self.model.addConstrs((self.available_capacity[t] == quicksum((1-self.
↪booking_variables[t,r])*self.room_capacity[r] for r in self.rooms) for t in␣
↪self.time), name='capacity_each_time_interval')

        # each booking can only book 1 room
        # self.model.addConstrs(quicksum(self.booking_req[i,r] for r in self.
↪rooms)<= 1 for i in self.booking_requests.index)

        # each booking cannot book below its capacity requirement
        self.model.addConstrs(self.booking_req[i,r] <= self.booking_requests.
↪loc[i, r] for i in self.booking_requests.index for r in self.rooms)

        # each booking should have 1 room booked
        self.model.addConstrs(quicksum(self.booking_req[i,r] for r in self.
↪rooms) == 1 for i in self.booking_requests.index)

        # booking can't overlap with existing booking
        # self.model.addConstrs(self.booking_req[i,r])

        # each booking made forces a change in the available capacity
        book_intervals  ={}
        for i in self.booking_requests.index:

            for r in self.rooms:
                interval = self.time_intervals(self.booking_requests.loc[i,␣
↪"start_time"], self.booking_requests.loc[i, "duration"])
                interval_sum = sum([self.booking_variables[t,r] for t in␣
↪interval])

                binary = self.model.addVar(vtype=GRB.BINARY, name='aux1')

                self.model.addConstr(interval_sum >= binary, name="aux1")

                self.model.addConstr(interval_sum <= 10e6*binary, name="aux2")
                self.model.addConstr(self.booking_req[i,r] <=  1-binary )

                book_intervals[(i,r)] = interval

        # avoid doubling up on bookings
        for k,v in book_intervals.items():

            # check each interval againts the other intervals
            for i,j in book_intervals.items():
                if i[1] == k[1] and i[0] != k[0]:
                    # print(i[1],k[1])
                    # print(i[0],k[0])
                    if len(set(v).intersection(set(j))) > 0:
                        # print(j)
                        # print(v)
                        self.model.addConstr(quicksum(self.booking_req[r, i[1]]␣
↪for r in self.booking_requests.index) <= 1, name = "aux3")
                        # self.model.addConstr(self.booking_req[i[0], i[1]]<=0,␣
↪name="aux3")
                        # self.model.addConstr(self.booking_req[k[0], k[1]]<=0)




    def time_intervals(self, start_time, duration):
        # create time intervals
        end_time = datetime.datetime.strptime(start_time, '%H:%M') + datetime.
↪timedelta(hours=duration)
        time_inverals = pd.date_range(start=datetime.datetime.
↪strptime(start_time, '%H:%M'), end=end_time, freq='15min').map(lambda x: x.
↪strftime("%H:%M"))
        return time_inverals.to_list()

    def get_solution(self):

        self.model.setObjective(quicksum(self.booking_req[i,r]*self.
↪room_capacity[r] for i in self.booking_requests.index for r in self.rooms ),␣
↪GRB.MINIMIZE)
        self.model.optimize()

        if self.model.status == GRB.Status.OPTIMAL:
            print("Best Allocation of requests: ")
            if len(self.booking_requests) > 0:

                y = self.model.getAttr('x', self.booking_req)
                for k,v in y.items():
                    if v > 0:
                        print(k,v)
            else:
                print("No requests to allocate")
        else:
            print("No solution found\n")
            print("Please drop a booking request, current number of bookings␣
↪can't all be served")



    def get_booking_variables(self):
        self.model.update()
        return self.model.getAttr('x', self.booking_variables)



    def get_constraints(self):
        self.model.update()
        return self.model.getConstrs()

    def reset_constraints(self):
        self.model.remove(self.model.getConstrs()[:])
```

Testing of the model below

[ ]:

```
test = BookingOptimiser(date_optimised = '2022-03-14', scheduling_table =␣
↪scheduling_table, booking_requests = booking_requests_rf)
```

[ ]:

```
test.create_constraints()
```

[ ]:

```
test.model.update()
test.model.getConstrByName("aux3")
```

[ ]: <gurobi.Constr aux3>

[ ]:

```
test.get_solution()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[arm])
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
Optimize a model with 3000 rows, 2364 columns and 10596 nonzeros
Model fingerprint: 0x319c74ab
Variable types: 0 continuous, 2364 integer (2268 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+07]
  Objective range  [4e+01, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+03]
```

```
Presolve removed 2756 rows and 2107 columns
Presolve time: 0.00s
Presolved: 244 rows, 257 columns, 940 nonzeros
Variable types: 0 continuous, 257 integer (257 binary)
Found heuristic solution: objective 434.0000000

Root relaxation: objective 4.020000e+02, 21 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     402.0000000  402.00000  0.00%     -    0s

Explored 1 nodes (21 simplex iterations) in 0.01 seconds (0.01 work units)
Thread count was 10 (of 10 available processors)

Solution count 2: 402 434

Optimal solution found (tolerance 1.00e-04)
Best objective 4.020000000000e+02, best bound 4.020000000000e+02, gap 0.0000%
Best Allocation of requests:
(3, 'LT5') 1.0
(4, 'LT10') 1.0
(5, 'LT12') 1.0
(6, 'LT9') 1.0
(7, 'Trans') 1.0
(8, 'LT3') 1.0
```

1.4 Results for given day

Use the widgets to pick the day to optimise. Only days that have booking requests recorded will be optimised.

[ ]:

```
# date picker
date_picker_2 = widgets.DatePicker(description='Date:', value=datetime.date.
  ↪today())

button = widgets.Button(description='Get Allocations', button_style='',␣
  ↪icon='check')
button_out = widgets.Output()


display(date_picker_2)
```

```
DatePicker(value=datetime.date(2022, 3, 14), description='Date:')
```

```
[ ]: # run optimisation

     date = date_picker_2.value
     # date to str
     date = date.strftime("%Y-%m-%d")
     print(date)
     model = BookingOptimiser(date_optimised = date, scheduling_table =␣
      ↪scheduling_table, booking_requests = booking_requests_rf)
     model.create_constraints()
     model.get_solution()
```

```
2022-03-15
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[arm])
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
Optimize a model with 2282 rows, 2196 columns and 5052 nonzeros
Model fingerprint: 0xce1e8f61
Variable types: 0 continuous, 2196 integer (2100 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+07]
  Objective range  [4e+01, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+03]
Presolve removed 2282 rows and 2196 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 10 available processors)

Solution count 1: 94

Optimal solution found (tolerance 1.00e-04)
Best objective 9.400000000000e+01, best bound 9.400000000000e+01, gap 0.0000%
Best Allocation of requests:
(1, 'LT5') 1.0
(2, 'LT5') 1.0
```

```
[ ]:
```