Code 2

April 1, 2024

[1]:
```
# Import the useful packages and install the Ortools
import numpy as np
import pandas as pd
%pip install Ortools
from ortools.sat.python import cp_model
```

```
Requirement already satisfied: Ortools in /opt/conda/lib/python3.9/site-packages
(9.9.3963)
Requirement already satisfied: protobuf>=4.25.3 in
/opt/conda/lib/python3.9/site-packages (from Ortools) (5.26.1)
Requirement already satisfied: absl-py>=2.0.0 in /opt/conda/lib/python3.9/site-
packages (from Ortools) (2.1.0)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.9/site-
packages (from Ortools) (1.23.5)
Requirement already satisfied: immutabledict>=3.0.0 in
/opt/conda/lib/python3.9/site-packages (from Ortools) (4.2.0)
Requirement already satisfied: pandas>=2.0.0 in /opt/conda/lib/python3.9/site-
packages (from Ortools) (2.1.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.9/site-packages (from pandas>=2.0.0->Ortools) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.9/site-
packages (from pandas>=2.0.0->Ortools) (2023.3)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.9/site-
packages (from pandas>=2.0.0->Ortools) (2022.4)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-
packages (from python-dateutil>=2.8.2->pandas>=2.0.0->Ortools) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
pip install openpyxl==3.1.0
```

```
Requirement already satisfied: openpyxl==3.1.0 in /opt/conda/lib/python3.9/site-
packages (3.1.0)
Requirement already satisfied: et-xmlfile in /opt/conda/lib/python3.9/site-
packages (from openpyxl==3.1.0) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

1 Preparation

1.1 Data cleaning

[3]:
```
# Import the dataset
df_filter = pd.read_excel("Filter.xlsx")

# Drop the columns we don't use.
df_drop = df_filter.drop(columns=['Total Hours', 'Activity','Zone␣
 ↪Name','Planned Size', 'Real Size'])

# Only consider Lectures
df_filter = df_drop[df_drop['Activity Type Name']=='*Lecture']
```

1.2 The code for finding the overlapping courses

[4]:
```
# Import the useful package
from datetime import datetime

class CourseSchedulerDF:
    def __init__(self, dataframe):
        # Make a copy of the input dataframe to avoid changing the original data
        self.dataframe = dataframe.copy()
        self.dataframe['Delivery Semester'] = self.dataframe['Delivery␣
 ↪Semester'].str.replace(r'\*', '', regex=True).str.strip()

    def _convert_time(self, hour_int):
        return str(hour_int)

    def find_overlapping_classes(self):
        # Initialize a list
        overlapping_classes = []

        # Double loop to compare each course with every other course for overlap
        for i in range(len(self.dataframe)):
            for j in range(i+1, len(self.dataframe)):
                course_i = self.dataframe.iloc[i]
                course_j = self.dataframe.iloc[j]

                # Convert the integer times to string
                start_i = self._convert_time(course_i["Scheduled Start Time"])
                end_i = self._convert_time(course_i["Scheduled End Time"])
                start_j = self._convert_time(course_j["Scheduled Start Time"])
                end_j = self._convert_time(course_j["Scheduled End Time"])

                # Check if times overlap, if the semester is the same, and if␣
 ↪the scheduled days are the same
```

```
                if (start_i < end_j and start_j < end_i) and \
                   (course_i["Delivery Semester"] == course_j["Delivery␣
↪Semester"]) and \
                   (course_i["Scheduled Days"] == course_j["Scheduled Days"]):

                    # If overlap is found, append the relevant information to␣
↪the list
                    overlapping_classes.append({
                        "Course 1": course_i["Course Code"],
                        "Course 2": course_j["Course Code"],
                        "Start Time Course 1": course_i["Scheduled Start Time"],
                        "End Time Course 1": course_i["Scheduled End Time"],
                        "Start Time Course 2": course_j["Scheduled Start Time"],
                        "End Time Course 2": course_j["Scheduled End Time"],
                        "Scheduled Days": course_i["Scheduled Days"],
                        "Delivery Semester": course_i["Delivery Semester"],
                        "Location Course 1": course_i["Allocated Location␣
↪Name"],
                        "Location Course 2": course_j["Allocated Location␣
↪Name"],
                    })

        # Return the results as a Dataframe
        return pd.DataFrame(overlapping_classes) if overlapping_classes else pd.
↪DataFrame(columns=[
            "Course 1", "Course 2",
            "Start Time Course 1", "End Time Course 1",
            "Start Time Course 2", "End Time Course 2",
            "Scheduled Days", "Delivery Semester","Location Course 1","Location␣
↪Course 2"
        ])
```

2 Level 8

[5]:
```
# Select the Level 8 courses from the dataset
df_L8 = df_filter[df_filter['Course Code'].str.contains('MATH08', regex=True)]

# Delete the repeat same course with the same schedule
df_L8 = df_L8.drop_duplicates(subset=['Course Name', 'Course Code', 'Scheduled␣
↪Days', 'Scheduled Start Time', 'Scheduled End Time'])

# Set the index
df_L8 = df_L8.reset_index(drop=True)
```

2.1 Original Clash for Level 8

[6]:
```
# Find the all clashing courses for Level 8
L8_scheduler = CourseSchedulerDF(df_L8)
L8_overlaps_df = L8_scheduler.find_overlapping_classes()
```

2.2 Apply CP model and reallocate all overlapping courses

[7]:
```
# Import the useful package
from datetime import timedelta

# Define the Constraint Programming model
model = cp_model.CpModel()

# Create variables for each course indicating their new start slots
course_slots = {}
for index, row in df_L8.iterrows():
    course_code = row['Course Code']
    # Assume each time slot represents an hour from 0 to 8

    ### Constraint 1: Limit course scheduling start time from 9:00 to 17:00
    course_slots[course_code] = model.NewIntVar(0, 8, 'slot_{}'.
 ↪format(course_code))


# Define a function to parse Scheduled Start Time into a time slot index
def get_slot_from_time(time_str):

    ### Constraint 1: Convert the start time to a time slot within the␣
 ↪operational hours
    return time_str - 9

for index, row in L8_overlaps_df.iterrows():
    course_1 = row['Course 1']
    course_2 = row['Course 2']

    # Get the time slot index for the courses
    slot_1 = get_slot_from_time(row['Start Time Course 1'])
    slot_2 = get_slot_from_time(row['Start Time Course 2'])

    # Ensure that the courses are not scheduled in the same time slot

    ### Constraint 2: Ensure all (compulsory) courses do not clash in Level 8
    model.Add(course_slots[course_1] != slot_1)
    model.Add(course_slots[course_2] != slot_2)

# Solve the model
```

```
solver = cp_model.CpSolver()
status = solver.Solve(model)

# Check the results
if status == cp_model.FEASIBLE or status == cp_model.OPTIMAL:

    # Create a dictionary to store course schedules by day and semester
    schedule_by_day_semester = {}

    # Iterate over the df_L8 DataFrame to organize courses by day and semester
    for index, row in df_L8.iterrows():
        course_code = row['Course Code']
        day = row['Scheduled Days']
        semester = row['Delivery Semester']
        start_time = row['Scheduled Start Time']
        end_time = row['Scheduled End Time']

        if (day, semester) in schedule_by_day_semester:
            # Add or update the course in the dictionary
            ### Constraint 3: Ensuring courses are assigned a single room␣
 ↪throughout the schedule
            schedule_by_day_semester[(day, semester)].setdefault(course_code,␣
 ↪{'Start Time': start_time, 'End Time': end_time})
        else:
            # Create a new entry in the dictionary for the new day and semester
            schedule_by_day_semester[(day, semester)] = {course_code: {'Start␣
 ↪Time': start_time, 'End Time': end_time}}

    # Create a list to store the final combined schedule
    L8_final_schedule = []

    for (day, semester), courses in schedule_by_day_semester.items():
        start_time = '9'  ### Constraint 1: Starting time is set to represent 9:
 ↪00
        for course_code, schedule in courses.items():
            duration = 1  # All durations in the dataset are 1
            # Get the location information for the corresponding course from␣
 ↪df_L8
            location = df_L8.loc[df_L8['Course Code'] == course_code,␣
 ↪'Allocated Location Name'].iloc[0]
            L8_final_schedule.append({
                'Course Code': course_code,
                'Scheduled Start Time': start_time,
                'Scheduled End Time': (datetime.strptime(start_time, "%H") +␣
 ↪timedelta(hours=duration)).strftime("%H"),
                'Scheduled Days': day,
```

```
                'Delivery Semester': semester,
                'Allocated Location Name': location
            })
            ### Constraint 2: Update the start time for the next course,␣
↪ensuring no overlap
            start_time = (datetime.strptime(start_time, "%H") +␣
↪timedelta(hours=duration)).strftime("%H")

    # Convert the final schedule list into a DataFrame
    L8_final_schedule_df = pd.DataFrame(L8_final_schedule)

    # Print or return the final DataFrame
    print(L8_final_schedule_df[['Course Code', 'Scheduled Start Time',␣
↪'Scheduled End Time', 'Scheduled Days', 'Delivery Semester', 'Allocated␣
↪Location Name']])

else:
    print("No solution found.")
```

	Course Code	Scheduled Start Time	Scheduled End Time	Scheduled Days
0	MATH08062	9	10	Monday
1	MATH08071	10	11	Monday
2	MATH08057	11	12	Monday
3	MATH08063	12	13	Monday
4	MATH08071	9	10	Thursday
5	MATH08074	10	11	Thursday
6	MATH08077	11	12	Thursday
7	MATH08057	12	13	Thursday
8	MATH08072	13	14	Thursday
9	MATH08066	14	15	Thursday
10	MATH08058	9	10	Thursday
11	MATH08075	10	11	Thursday
12	MATH08068	11	12	Thursday
13	MATH08059	12	13	Thursday
14	MATH08051	13	14	Thursday
15	MATH08058	9	10	Monday
16	MATH08051	10	11	Monday
17	MATH08065	9	10	Wednesday
18	MATH08064	10	11	Wednesday
19	MATH08074	9	10	Tuesday
20	MATH08072	10	11	Tuesday
21	MATH08066	11	12	Tuesday
22	MATH08063	12	13	Tuesday
23	MATH08075	9	10	Friday
24	MATH08063	10	11	Friday
25	MATH08075	9	10	Tuesday
26	MATH08064	10	11	Tuesday

```
27   MATH08059                    9                 10         Friday
28   MATH08063                    9                 10      Wednesday

   Delivery Semester              Allocated Location Name
0            * SEM 1               JCMB_Lecture Theatre C
1            * SEM 1                      JBB_Theatre 100
2            * SEM 1  GALT_ Gordon Aikman Lecture Theatre
3            * SEM 1             SB_Main Lecture Theatre
4            * SEM 1                      JBB_Theatre 100
5            * SEM 1  GALT_ Gordon Aikman Lecture Theatre
6            * SEM 1                AT_Lecture Theatre 5
7            * SEM 1  GALT_ Gordon Aikman Lecture Theatre
8            * SEM 1                AT_Lecture Theatre 2
9            * SEM 1               ASH_Lecture Theatre 1
10           * SEM 2  GALT_ Gordon Aikman Lecture Theatre
11           * SEM 2       NUC_1.14 - Oak Lecture Theatre
12           * SEM 2             SB_Main Lecture Theatre
13           * SEM 2               40GS_Lecture Theatre C
14           * SEM 2       NUC_1.14 - Oak Lecture Theatre
15           * SEM 2  GALT_ Gordon Aikman Lecture Theatre
16           * SEM 2       NUC_1.14 - Oak Lecture Theatre
17           * SEM 2       NUC_1.14 - Oak Lecture Theatre
18           * SEM 2             SB_Main Lecture Theatre
19           * SEM 1  GALT_ Gordon Aikman Lecture Theatre
20           * SEM 1                AT_Lecture Theatre 2
21           * SEM 1               ASH_Lecture Theatre 1
22           * SEM 1             SB_Main Lecture Theatre
23           * SEM 1       NUC_1.14 - Oak Lecture Theatre
24           * SEM 1             SB_Main Lecture Theatre
25           * SEM 2       NUC_1.14 - Oak Lecture Theatre
26           * SEM 2             SB_Main Lecture Theatre
27           * SEM 2               40GS_Lecture Theatre C
28           * SEM 1             SB_Main Lecture Theatre
```

2.3 Verify that there is no overlapping courses after reallocatng

[8]:
```
# Verify that there are no overlapping courses after rescheduling.
L8_final_scheduler = CourseSchedulerDF(L8_final_schedule_df)
L8_final_overlaps_df = L8_final_scheduler.find_overlapping_classes()

# Print the DataFrame
print(L8_final_overlaps_df)
```

```
Empty DataFrame
Columns: [Course 1, Course 2, Start Time Course 1, End Time Course 1, Start Time
Course 2, End Time Course 2, Scheduled Days, Delivery Semester, Location Course
1, Location Course 2]
Index: []
```

3 Level 10 + 11

[9]:
```
# Select Level 10+11 courses from the dataset
df_L1011 = df_filter[df_filter['Course Code'].str.contains('MATH10|MATH11',␣
 ↪regex=True)]

# Delete the repeat same course with the same schedule
df_L1011 = df_L1011.drop_duplicates(subset=['Course Name', 'Course Code',␣
 ↪'Scheduled Days', 'Scheduled Start Time', 'Scheduled End Time'])

# Set the index
df_L1011 = df_L1011.reset_index(drop=True)
```

In the dataset we have a column called 'Compulsory' where Y denotes a compulsory course for undergraduates, S denotes a compulsory course for postgraduates, YS denotes a compulsory course for both undergraduates and postgraduates, and N denotes an optional course.

3.1 Compulsory courses

[10]:
```
# Selet all compulsory courses in Level 10 and 11 (Y, YS, S)
df_compulsory = df_L1011[df_L1011['Compulsory'].str.contains('Y| YS |S',␣
 ↪regex=True)]
df_compulsory = df_compulsory.reset_index(drop=True)
```

3.1.1 Original Clash for compulsory courses

[11]:
```
# Find the all clashing courses for compulsory courses in Level 10+11
compulsory_scheduler = CourseSchedulerDF(df_compulsory)
compulsory_overlaps_df = compulsory_scheduler.find_overlapping_classes()
```

3.1.2 Apply CP model and reallocate all overlapping courses

[12]:
```
# Import the useful package
from datetime import timedelta

# Define the Constraint Programming model
model = cp_model.CpModel()

# Create variables for each course indicating their new start slots
course_slots = {}
for index, row in df_compulsory.iterrows():
    course_code = row['Course Code']
    # Assume each time slot represents an hour from 0 to 8

    ### Constraint 1: Limit course scheduling start time from 9:00 to 17:00
```

```
    course_slots[course_code] = model.NewIntVar(0, 8, 'slot_{}'.
 ↪format(course_code))


# Define a function to parse Scheduled Start Time into a time slot index
def get_slot_from_time(time_str):

    ### Constraint 1: Convert the start time to a time slot within the␣
 ↪operational hours
    return time_str - 9

for index, row in compulsory_overlaps_df.iterrows():
    course_1 = row['Course 1']
    course_2 = row['Course 2']

    # Get the time slot index for the courses
    slot_1 = get_slot_from_time(row['Start Time Course 1'])
    slot_2 = get_slot_from_time(row['Start Time Course 2'])

    # Ensure that the courses are not scheduled in the same time slot

    ### Constraint 2: Ensure all compulsory courses do not clash in Level 10+11
    model.Add(course_slots[course_1] != slot_1)
    model.Add(course_slots[course_2] != slot_2)

# Solve the model
solver = cp_model.CpSolver()
status = solver.Solve(model)

# Check the results
if status == cp_model.FEASIBLE or status == cp_model.OPTIMAL:

    # Create a dictionary to store course schedules by day and semester
    schedule_by_day_semester = {}

    # Iterate over the df_compulsory DataFrame to organize courses by day and␣
 ↪semester
    for index, row in df_compulsory.iterrows():
        course_code = row['Course Code']
        day = row['Scheduled Days']
        semester = row['Delivery Semester']
        start_time = row['Scheduled Start Time']
        end_time = row['Scheduled End Time']

        if (day, semester) in schedule_by_day_semester:
            # Add or update the course in the dictionary
```

```
            ### Constraint 3: Ensuring courses are assigned a single room␣
↪throughout the schedule
            schedule_by_day_semester[(day, semester)].setdefault(course_code,␣
↪{'Start Time': start_time, 'End Time': end_time})
        else:
            # Create a new entry in the dictionary for the new day and semester
            schedule_by_day_semester[(day, semester)] = {course_code: {'Start␣
↪Time': start_time, 'End Time': end_time}}

    # Create a list to store the final combined schedule
    compulsory_final_schedule = []

    for (day, semester), courses in schedule_by_day_semester.items():
        start_time = '9'  ### Constraint 1: Starting time is set to represent 9:
↪00
        for course_code, schedule in courses.items():
            duration = 1  # All durations in the dataset are 1
            # Get the location information for the corresponding course from␣
↪df_compulsory
            location = df_compulsory.loc[df_compulsory['Course Code'] ==␣
↪course_code, 'Allocated Location Name'].iloc[0]
            compulsory_final_schedule.append({
                'Course Code': course_code,
                'Scheduled Start Time': start_time,
                'Scheduled End Time': (datetime.strptime(start_time, "%H") +␣
↪timedelta(hours=duration)).strftime("%H"),
                'Scheduled Days': day,
                'Delivery Semester': semester,
                'Allocated Location Name': location
            })
            ### Constraint 2: Update the start time for the next course,␣
↪ensuring no overlap
            start_time = (datetime.strptime(start_time, "%H") +␣
↪timedelta(hours=duration)).strftime("%H")

    # Convert the final schedule list into a DataFrame
    compulsory_final_schedule_df = pd.DataFrame(compulsory_final_schedule)

    # Print or return the final DataFrame
    print(compulsory_final_schedule_df[['Course Code', 'Scheduled Start Time',␣
↪'Scheduled End Time', 'Scheduled Days', 'Delivery Semester', 'Allocated␣
↪Location Name']])

else:
    print("No solution found.")
```

	Course Code	Scheduled Start Time	Scheduled End Time	Scheduled Days
0	MATH11140	9	10	Friday
1	MATH11175	10	11	Friday
2	MATH10067	11	12	Friday
3	MATH11158	12	13	Friday
4	MATH11140	9	10	Tuesday
5	MATH11202	10	11	Tuesday
6	MATH11028	11	12	Tuesday
7	MATH10093	12	13	Tuesday
8	MATH11150	13	14	Tuesday
9	MATH11177	9	10	Tuesday
10	MATH10098	10	11	Tuesday
11	MATH10095	11	12	Tuesday
12	MATH10007	12	13	Tuesday
13	MATH10065	9	10	Monday
14	MATH10068	10	11	Monday
15	MATH10066	11	12	Monday
16	MATH10013	12	13	Monday
17	MATH11007	13	14	Monday
18	MATH11199	14	15	Monday
19	MATH11176	15	16	Monday
20	MATH11154	16	17	Monday
21	MATH11111	9	10	Thursday
22	MATH10068	10	11	Thursday
23	MATH10066	11	12	Thursday
24	MATH11154	12	13	Thursday
25	MATH11187	9	10	Wednesday
26	MATH10013	10	11	Wednesday
27	MATH11197	11	12	Wednesday
28	MATH10007	12	13	Wednesday
29	MATH10069	9	10	Monday
30	MATH11185	10	11	Monday
31	MATH11181	11	12	Monday
32	MATH10060	12	13	Monday
33	MATH11207	13	14	Monday
34	MATH11197	14	15	Monday
35	MATH10083	15	16	Monday
36	MATH10069	9	10	Thursday
37	MATH10067	10	11	Thursday
38	MATH11185	11	12	Thursday
39	MATH11207	12	13	Thursday
40	MATH11197	13	14	Thursday
41	MATH11188	14	15	Thursday
42	MATH10066	9	10	Friday
43	MATH10098	10	11	Friday
44	MATH10095	11	12	Friday
45	MATH11157	9	10	Wednesday

	Delivery Semester	Allocated Location Name
0	* SEM 2	HBB_Lecture Theatre 2
1	* SEM 2	NUC_B.01 - Alder Lecture Theatre
2	* SEM 2	NUC_1.14 - Oak Lecture Theatre
3	* SEM 2	JCMB_Lecture Theatre A
4	* SEM 2	HBB_Lecture Theatre 2
5	* SEM 2	JCMB_1501
6	* SEM 2	JCMB_Lecture Theatre C
7	* SEM 2	JCMB_Lecture Theatre A
8	* SEM 2	JCMB_5326
9	* SEM 1	JCMB_Lecture Theatre A
10	* SEM 1	JCMB_Lecture Theatre B
11	* SEM 1	ASH_Lecture Theatre 1
12	* SEM 1	ASH_Lecture Theatre 1
13	* SEM 1	JCMB_Lecture Theatre A
14	* SEM 1	SB_Main Lecture Theatre
15	* SEM 1	SB_Main Lecture Theatre
16	* SEM 1	JCMB_Lecture Theatre A
17	* SEM 1	JCMB_Lecture Theatre C
18	* SEM 1	SB_Main Lecture Theatre
19	* SEM 1	JCMB_Lecture Theatre A
20	* SEM 1	MH_G.26 - Charlotte Murchison Lecture Theatre
21	* SEM 1	JBB_Theatre 250
22	* SEM 1	SB_Main Lecture Theatre
23	* SEM 1	SB_Main Lecture Theatre
24	* SEM 1	MH_G.26 - Charlotte Murchison Lecture Theatre
25	* SEM 1	JCMB_Lecture Theatre B
26	* SEM 1	JCMB_Lecture Theatre A
27	* SEM 1	JCMB_5327
28	* SEM 1	ASH_Lecture Theatre 1
29	* SEM 2	NUC_1.14 - Oak Lecture Theatre
30	* SEM 2	JCMB_Lecture Theatre A
31	* SEM 2	JCMB_Lecture Theatre C
32	* SEM 2	JCMB_Lecture Theatre A
33	* SEM 2	JCMB_Lecture Theatre C
34	* SEM 2	JCMB_5327
35	* SEM 2	JCMB_5328
36	* SEM 2	NUC_1.14 - Oak Lecture Theatre
37	* SEM 2	NUC_1.14 - Oak Lecture Theatre
38	* SEM 2	JCMB_Lecture Theatre A
39	* SEM 2	JCMB_Lecture Theatre C
40	* SEM 2	JCMB_5327
41	* SEM 2	JCMB_Lecture Theatre B
42	* SEM 1	SB_Main Lecture Theatre
43	* SEM 1	JCMB_Lecture Theatre B
44	* SEM 1	ASH_Lecture Theatre 1
45	* SEM 2	JCMB_5327

3.1.3 Verify that there is no overlapping courses after reallocatng

[13]:
```
# # Verify that there are no overlapping courses after rescheduling.
compulsory_final_scheduler = CourseSchedulerDF(compulsory_final_schedule_df)
compulsory_final_overlaps_df = compulsory_final_scheduler.
 ↪find_overlapping_classes()

# Print the DataFrame
print(compulsory_final_overlaps_df)
```

```
Empty DataFrame
Columns: [Course 1, Course 2, Start Time Course 1, End Time Course 1, Start Time
Course 2, End Time Course 2, Scheduled Days, Delivery Semester, Location Course
1, Location Course 2]
Index: []
```

3.2 Non-compulsory (optional) courses

[14]:
```
# Selet all optional courses in Level 10 and 11
df_non_compulsory = df_L1011[df_L1011['Compulsory'].str.contains('N',␣
 ↪regex=True)]
df_non_compulsory = df_non_compulsory.reset_index(drop=True)
```

3.2.1 Original Clash for non-compulsory (optional) courses

[15]:
```
# Find all overlapping courses of optional courses in Level 10+11
non_compulsory_scheduler = CourseSchedulerDF(df_non_compulsory)
non_compulsory_overlaps_df = non_compulsory_scheduler.find_overlapping_classes()
```

3.2.2 Apply CP model and reallocate all overlapping courses

[16]:
```
# Import the useful package
from datetime import timedelta

# Define the Constraint Programming model
model = cp_model.CpModel()

# Create variables for each course indicating their new start slots
course_slots = {}
for index, row in df_non_compulsory.iterrows():
    course_code = row['Course Code']
    # Assume each time slot represents an hour from 0 to 8

    ### Constraint 1: Limit course scheduling start time from 9:00 to 17:00
    course_slots[course_code] = model.NewIntVar(0, 8, 'slot_{}'.
 ↪format(course_code))
```

```
# Define a function to parse Scheduled Start Time into a time slot index
def get_slot_from_time(time_str):

    ### Constraint 1: Convert the start time to a time slot within the␣
 ↪operational hours
    return time_str - 9

for index, row in non_compulsory_overlaps_df.iterrows():
    course_1 = row['Course 1']
    course_2 = row['Course 2']

    # Get the time slot index for the courses
    slot_1 = get_slot_from_time(row['Start Time Course 1'])
    slot_2 = get_slot_from_time(row['Start Time Course 2'])

    # Ensure that the courses are not scheduled in the same time slot

    ### Constraint 2: Ensure all optional courses do not clash in Level 10+11
    model.Add(course_slots[course_1] != slot_1)
    model.Add(course_slots[course_2] != slot_2)

# Solve the model
solver = cp_model.CpSolver()
status = solver.Solve(model)

# Check the results
if status == cp_model.FEASIBLE or status == cp_model.OPTIMAL:

    # Create a dictionary to store course schedules by day and semester
    schedule_by_day_semester = {}

    # Iterate over the df_non_compulsory DataFrame to organize courses by day␣
 ↪and semester
    for index, row in df_non_compulsory.iterrows():
        course_code = row['Course Code']
        day = row['Scheduled Days']
        semester = row['Delivery Semester']
        start_time = row['Scheduled Start Time']
        end_time = row['Scheduled End Time']

        if (day, semester) in schedule_by_day_semester:
            # Add or update the course in the dictionary
            ### Constraint 3: Ensuring courses are assigned a single room␣
 ↪throughout the schedule
            schedule_by_day_semester[(day, semester)].setdefault(course_code,␣
 ↪{'Start Time': start_time, 'End Time': end_time})
```

```
        else:
            # Create a new entry in the dictionary for the new day and semester
            schedule_by_day_semester[(day, semester)] = {course_code: {'Start␣
↪Time': start_time, 'End Time': end_time}}

    # Create a list to store the final combined schedule
    non_compulsory_final_schedule = []

    for (day, semester), courses in schedule_by_day_semester.items():
        start_time = '9'  ### Constraint 1: Starting time is set to represent 9:
↪00
        for course_code, schedule in courses.items():
            duration = 1  # All durations in the dataset are 1
            # Get the location information for the corresponding course from␣
↪df_non_compulsory
            location = df_non_compulsory.loc[df_non_compulsory['Course Code']␣
↪== course_code, 'Allocated Location Name'].iloc[0]
            non_compulsory_final_schedule.append({
                'Course Code': course_code,
                'Scheduled Start Time': start_time,
                'Scheduled End Time': (datetime.strptime(start_time, "%H") +␣
↪timedelta(hours=duration)).strftime("%H"),
                'Scheduled Days': day,
                'Delivery Semester': semester,
                'Allocated Location Name': location
            })
            ### Constraint 2: Update the start time for the next course,␣
↪ensuring no overlap
            start_time = (datetime.strptime(start_time, "%H") +␣
↪timedelta(hours=duration)).strftime("%H")

    # Convert the final schedule list into a DataFrame
    non_compulsory_final_schedule_df = pd.
↪DataFrame(non_compulsory_final_schedule)

    # Print or return the final DataFrame
    print(non_compulsory_final_schedule_df[['Course Code', 'Scheduled Start␣
↪Time', 'Scheduled End Time', 'Scheduled Days', 'Delivery Semester',␣
↪'Allocated Location Name']])

else:
    print("No solution found.")
```

```
   Course Code Scheduled Start Time Scheduled End Time Scheduled Days  \
0    MATH10086                    9                  10         Friday
1    MATH11138                   10                  11         Friday
2    MATH10071                   11                  12         Friday
```

```
3    MATH10101                   12                  13          Friday
4    MATH10086                    9                  10        Thursday
..         …                     …                   …           …
78   MATH10028                   18                  19        Thursday
79   MATH11144                   19                  20        Thursday
80   MATH11179                   20                  21        Thursday
81   MATH10024                    9                  10          Monday
82   MATH10024                    9                  10        Thursday

    Delivery Semester  Allocated Location Name
0              * SEM 2   JCMB_Lecture Theatre C
1              * SEM 2                JCMB_5327
2              * SEM 2  SB_Main Lecture Theatre
3              * SEM 2   JCMB_Lecture Theatre B
4              * SEM 2   JCMB_Lecture Theatre C
..                 …                      …
78             * SEM 1                JCMB_5328
79             * SEM 1                JCMB_5326
80             * SEM 1   JCMB_Lecture Theatre A
81  * SEM 1 + * SEM 2  SB_Main Lecture Theatre
82  * SEM 1 + * SEM 2  SB_Main Lecture Theatre

[83 rows x 6 columns]
```

3.2.3 Reallocated courses of Scheduled Start Time greater than 17 (not satisfied Constraint 1)

[17]:
```
# Categorise courses by Scheduled Start Time
non_compulsory_final_schedule_df['Scheduled Start Time'] = pd.
 ↪to_numeric(non_compulsory_final_schedule_df['Scheduled Start Time'])

# Select Scheduled Start Time < 18
early_courses_df =␣
 ↪non_compulsory_final_schedule_df[non_compulsory_final_schedule_df['Scheduled␣
 ↪Start Time'] < 18]

# Select Scheduled Start Time > 17, which is not satisfied our Constraint 1
late_courses_df =␣
 ↪non_compulsory_final_schedule_df[non_compulsory_final_schedule_df['Scheduled␣
 ↪Start Time'] > 17]
```

[18]:
```
# Reallocate the randomly in 9:00-17:00
late_courses_df = late_courses_df.copy()
np.random.seed(42)
unique_days = late_courses_df['Scheduled Days'].unique()

# Randomizing the 'Scheduled Start Time' within 9-17
```

```
late_courses_df.loc[:, 'Scheduled Start Time'] = np.random.randint(9, 18,␣
 ↪size=len(late_courses_df))

# Setting 'Scheduled End Time' as 'Scheduled Start Time' + 1
late_courses_df.loc[:, 'Scheduled End Time'] = late_courses_df.loc[:,␣
 ↪'Scheduled Start Time'] + 1

# Randomizing 'Scheduled Days'
late_courses_df.loc[:, 'Scheduled Days'] = np.random.choice(unique_days,␣
 ↪size=len(late_courses_df))
```

```
/tmp/ipykernel_355/3860324715.py:10: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  late_courses_df.loc[:, 'Scheduled End Time'] = late_courses_df.loc[:,
'Scheduled Start Time'] + 1
```

3.2.4 Form the final optional courses schedule

[19]:
```
# Merge the new late courses schedule and early course schedule to form the␣
 ↪optional courses timetable
merged_df = pd.concat([late_courses_df, early_courses_df], ignore_index=True)
```

3.2.5 Overlapping courses of final optional courses schedule

[20]:
```
# Find the all overlapping courses of final optional courses schedule
merged_final_scheduler = CourseSchedulerDF(merged_df)
merged_final_overlaps_df = merged_final_scheduler.find_overlapping_classes()
```

4 Combine the timetable of Level 8, Level 10+11 compulsory courses and Level 10+11 optional courses

[21]:
```
# Combine the schedules into one DataFrame
combined_final_schedule_df = pd.concat([L8_final_schedule_df,␣
 ↪compulsory_final_schedule_df, merged_df], ignore_index=True)
```

4.1 Overlapping courses of combined timetable

[22]:
```
# Find the overlapping courses of combined timetable
combined_final_scheduler = CourseSchedulerDF(combined_final_schedule_df)
combined_final_overlaps_df = combined_final_scheduler.find_overlapping_classes()

# Print the DataFrame
```

```
combined_final_overlaps_df
```

[22]:
```
     Course 1   Course 2 Start Time Course 1 End Time Course 1  \
0   MATH08071  MATH10068                  10                11
1   MATH08071  MATH10053                  10                11
2   MATH08057  MATH10066                  11                12
3   MATH08057  MATH11236                  11                12
4   MATH08063  MATH10013                  12                13
..        …        …                    …                 …
83  MATH10100  MATH11179                  16                17
84  MATH10100  MATH11220                  16                17
85  MATH11144  MATH10079                  13                14
86  MATH10028  MATH10076                  12                13
87  MATH11179  MATH11220                  16                17

   Start Time Course 2 End Time Course 2 Scheduled Days Delivery Semester  \
0                   10                11         Monday             SEM 1
1                   10                11         Monday             SEM 1
2                   11                12         Monday             SEM 1
3                   11                12         Monday             SEM 1
4                   12                13         Monday             SEM 1
..                 …                 …             …                …
83                  16                17       Thursday             SEM 1
84                  16                17       Thursday             SEM 1
85                  13                14       Thursday             SEM 1
86                  12                13        Tuesday             SEM 1
87                  16                17       Thursday             SEM 1

                    Location Course 1        Location Course 2
0                       JBB_Theatre 100  SB_Main Lecture Theatre
1                       JBB_Theatre 100          JBB_Theatre 100
2   GALT_ Gordon Aikman Lecture Theatre  SB_Main Lecture Theatre
3   GALT_ Gordon Aikman Lecture Theatre                JCMB_5328
4               SB_Main Lecture Theatre   JCMB_Lecture Theatre A
..                                  …                       …
83                           JCMB_5327   JCMB_Lecture Theatre A
84                           JCMB_5327               7-8CS_1.01
85                           JCMB_5326                JCMB_5327
86                           JCMB_5328                JCMB_1501
87               JCMB_Lecture Theatre A               7-8CS_1.01

[88 rows x 10 columns]
```

[23]:
```
# Find the same location courses
same_location_df =␣
 ↪combined_final_overlaps_df[combined_final_overlaps_df['Location Course 1']␣
 ↪== combined_final_overlaps_df['Location Course 2']]
```

```
same_location_df
```

[23]:

	Course 1	Course 2	Start Time Course 1	End Time Course 1
1	MATH08071	MATH10053	10	11
15	MATH08075	MATH10067	10	11
28	MATH08066	MATH10095	11	12
45	MATH11150	MATH11193	13	14
67	MATH11197	MATH11233	14	15
78	MATH11183	MATH11147	12	13
82	MATH11231	MATH10099	11	12

	Start Time Course 2	End Time Course 2	Scheduled Days	Delivery Semester
1	10	11	Monday	SEM 1
15	10	11	Thursday	SEM 2
28	11	12	Tuesday	SEM 1
45	13	14	Tuesday	SEM 2
67	14	15	Monday	SEM 2
78	12	13	Tuesday	SEM 2
82	11	12	Tuesday	SEM 1

	Location Course 1	Location Course 2
1	JBB_Theatre 100	JBB_Theatre 100
15	NUC_1.14 - Oak Lecture Theatre	NUC_1.14 - Oak Lecture Theatre
28	ASH_Lecture Theatre 1	ASH_Lecture Theatre 1
45	JCMB_5326	JCMB_5326
67	JCMB_5327	JCMB_5327
78	JCMB_Lecture Theatre C	JCMB_Lecture Theatre C
82	JCMB_5327	JCMB_5327

4.2 Fake Room

Size:

- Fake Room 1: 100
- Fake Room 2: 400

Course Code:

- Fake Room 1: MATH10053,MATH10099, MATH11193, MATH11233, MATH11147
- Fake Room 2: MATH10067, MATH10095

[24]:
```
# Define the course codes for which the locations need to be updated to 'Fake␣
 ↪Room 1 and 2'.
course_codes_fake_room_1 = ['MATH10053', 'MATH10099', 'MATH11193', 'MATH11233',␣
 ↪'MATH11147']
course_codes_fake_room_2 = ['MATH10067', 'MATH10095']

# Update 'Allocated Location Name' for the 'Course Code' 'MATH10099' to 'Fake␣
 ↪Room 1'.
```

```
combined_final_schedule_df.loc[combined_final_schedule_df['Course Code'].
 ↪isin(course_codes_fake_room_1), 'Allocated Location Name'] = 'Fake Room 1'

# Update 'Allocated Location Name' for the specified 'Course Codes' to 'Fake␣
 ↪Room 2'.
combined_final_schedule_df.loc[combined_final_schedule_df['Course Code'].
 ↪isin(course_codes_fake_room_2), 'Allocated Location Name'] = 'Fake Room 2'
```

5 Final Reallocated Timetable

[25]:
```
# Generate a Excel file of final timetable
Final_file_name = 'Final Timetable.xlsx'
combined_final_schedule_df.to_excel(Final_file_name, index=False)

# Print the final timetable
combined_final_schedule_df
```

[25]:

	Course Code	Scheduled Start Time	Scheduled End Time	Scheduled Days	Delivery Semester	Allocated Location Name
0	MATH08062	9	10	Monday	* SEM 1	JCMB_Lecture Theatre C
1	MATH08071	10	11	Monday	* SEM 1	JBB_Theatre 100
2	MATH08057	11	12	Monday	* SEM 1	GALT_ Gordon Aikman Lecture Theatre
3	MATH08063	12	13	Monday	* SEM 1	SB_Main Lecture Theatre
4	MATH08071	9	10	Thursday	* SEM 1	JBB_Theatre 100
..	…	…	…	…	…	…
153	MATH10100	15	16	Thursday	* SEM 1	JCMB_5327
154	MATH11220	16	17	Thursday	* SEM 1	7-8CS_1.01
155	MATH10102	17	18	Thursday	* SEM 1	JCMB_5327
156	MATH10024	9	10	Monday	* SEM 1 + * SEM 2	SB_Main Lecture Theatre
157	MATH10024	9	10	Thursday	* SEM 1 + * SEM 2	SB_Main Lecture Theatre

[158 rows x 6 columns]

6 Comparison: all overlapping courses of original timetable

[26]:
```
# Find the all overlapping courses
filter_scheduler = CourseSchedulerDF(df_filter)
filter_overlaps_df = filter_scheduler.find_overlapping_classes()

# Print the DataFrame
filter_overlaps_df
```

[26]:

	Course 1	Course 2	Start Time Course 1	End Time Course 1	\
0	MATH08062	MATH10053	14	15	
1	MATH08062	MATH10065	14	15	
2	MATH08062	MATH10074	14	15	
3	MATH08062	MATH10065	15	16	
4	MATH08071	MATH10072	10	11	
..	…	…	…	…	
210	MATH08051	MATH11227	14	15	
211	MATH11154	MATH11179	15	16	
212	MATH11150	MATH11183	14	15	
213	MATH10083	MATH11227	14	15	
214	MATH10083	MATH11227	15	16	

	Start Time Course 2	End Time Course 2	Scheduled Days	Delivery Semester	\
0	14	15	Monday	SEM 1	
1	14	15	Monday	SEM 1	
2	14	15	Monday	SEM 1	
3	15	16	Monday	SEM 1	
4	10	11	Monday	SEM 1	
..	…	…	…	…	
210	14	15	Monday	SEM 2	
211	15	16	Thursday	SEM 1	
212	14	15	Tuesday	SEM 2	
213	14	15	Monday	SEM 2	
214	15	16	Monday	SEM 2	

	Location Course 1	\
0	JCMB_Lecture Theatre C	
1	JCMB_Lecture Theatre C	
2	JCMB_Lecture Theatre C	
3	JCMB_Lecture Theatre C	
4	JBB_Theatre 100	
..	…	
210	NUC_1.14 - Oak Lecture Theatre	
211	JCMB_Lecture Theatre B	
212	JCMB_5326	
213	JCMB_5328	
214	JCMB_5328	

```
                                  Location Course 2
0                                   JBB_Theatre 100
1                             JCMB_Lecture Theatre A
2                             JCMB_Lecture Theatre B
3                             JCMB_Lecture Theatre A
4    MH_G.26 - Charlotte Murchison Lecture Theatre
..                                                ...
210                                       JCMB_5326
211                           JCMB_Lecture Theatre A
212                           JCMB_Lecture Theatre C
213                                       JCMB_5326
214                                       JCMB_5326

[215 rows x 10 columns]
```

7 All overlapping courses of final reallocated timetable

[27]:
```
# Find the all overlapping courses of final timetable
final_scheduler = CourseSchedulerDF(combined_final_schedule_df)
final_overlaps_df = final_scheduler.find_overlapping_classes()

# Generate a Excel file of clashing courses for final timetable
clashes_file_name = 'Final Clashes.xlsx'
final_overlaps_df.to_excel(clashes_file_name, index=False)

# Print the all overlapping courses of final timetable
final_overlaps_df
```

[27]:
```
     Course 1   Course 2 Start Time Course 1 End Time Course 1  \
0   MATH08071  MATH10068                  10                11
1   MATH08071  MATH10053                  10                11
2   MATH08057  MATH10066                  11                12
3   MATH08057  MATH11236                  11                12
4   MATH08063  MATH10013                  12                13
..        ...        ...                 ...               ...
83  MATH10100  MATH11179                  16                17
84  MATH10100  MATH11220                  16                17
85  MATH11144  MATH10079                  13                14
86  MATH10028  MATH10076                  12                13
87  MATH11179  MATH11220                  16                17

   Start Time Course 2 End Time Course 2 Scheduled Days Delivery Semester  \
0                   10                11         Monday             SEM 1
1                   10                11         Monday             SEM 1
2                   11                12         Monday             SEM 1
```

```
3                    11                 12          Monday              SEM 1
4                    12                 13          Monday              SEM 1
..                  …                  …            …                   …
83                   16                 17        Thursday              SEM 1
84                   16                 17        Thursday              SEM 1
85                   13                 14        Thursday              SEM 1
86                   12                 13         Tuesday              SEM 1
87                   16                 17        Thursday              SEM 1

                      Location Course 1        Location Course 2
0                       JBB_Theatre 100  SB_Main Lecture Theatre
1                       JBB_Theatre 100              Fake Room 1
2   GALT_ Gordon Aikman Lecture Theatre  SB_Main Lecture Theatre
3   GALT_ Gordon Aikman Lecture Theatre                JCMB_5328
4               SB_Main Lecture Theatre   JCMB_Lecture Theatre A
..                                   …                       …
83                            JCMB_5327   JCMB_Lecture Theatre A
84                            JCMB_5327              7-8CS_1.01
85                            JCMB_5326               JCMB_5327
86                            JCMB_5328               JCMB_1501
87               JCMB_Lecture Theatre A               7-8CS_1.01

[88 rows x 10 columns]
```

8 Validation above code

[28]:
```python
course_codes = [
    "MATH11177", "MATH11119", "MATH10065", "MATH11111", "MATH11187",
    "MATH11192", "MATH11007", "MATH10064", "MATH11190", "MATH11028",
    "MATH11176", "MATH11188", "MATH11131"
]

# Filter the rows where the 'Course Code' is in the list of course codes to␣
 ↪filter
test_df = df_filter[df_filter['Course Code'].isin(course_codes)]

# Find the overlapping courses
test_scheduler = CourseSchedulerDF(test_df)
test_overlaps_df =test_scheduler.find_overlapping_classes()

# Check if there are any overlaps
if not test_overlaps_df.empty:
    print(test_overlaps_df)
else:
    print("Congratulations! You don't have overlapping courses!")
```

```
    Course 1   Course 2  Start Time Course 1  End Time Course 1  \
0  MATH11192  MATH11131                   10                 11
1  MATH10064  MATH11190                   13                 14

   Start Time Course 2  End Time Course 2 Scheduled Days Delivery Semester  \
0                   10                 11      Wednesday             SEM 2
1                   13                 14       Thursday             SEM 2

        Location Course 1       Location Course 2
0  JCMB_Lecture Theatre B  JCMB_Lecture Theatre C
1  JCMB_Lecture Theatre B               JCMB_1501
```

9 Apply the model

[29]:
```
# Filter the rows where the 'Course Code' is in the list of course codes to␣
 ↪filter
test2_df = combined_final_schedule_df[combined_final_schedule_df['Course Code'].
 ↪isin(course_codes)]
test2_scheduler = CourseSchedulerDF(test2_df)
test2_overlaps_df =test2_scheduler.find_overlapping_classes()
if not test2_overlaps_df.empty:
    print(test2_overlaps_df)
else:
    print("Congratulations! You don't have overlapping courses!")
```

```
Congratulations! You don't have overlapping courses!
```