Prescriptive Analytics (Optimization) Project:

University Course Timetabling

Boyan Li, Vivi Li, Jennifer Liu, Alisa Liu, Ko-Jen Wang

MGSC 662: Decision Analytics - Fall 2023

Prof. Javad Nasiry

Date: November 23, 2023

1 Introduction

Creating a university course timetable is a task that often goes unnoticed, yet it is a cornerstone of academic life. This process is not just about plotting lessons on a grid; it's a complex puzzle, a fine art of balancing diverse needs and constraints to create a harmonious educational experience. At its core, the university course timetable is a meticulous coordination of resources. Every slot in this timetable carries the weight of multiple considerations: the strategic timing of classes, thoughtfully arranged to coincide with the optimal periods for learning and teaching, considering the varied rhythms of the academic day; the challenge of room assignments, where each space has its unique attributes and limitations; and the diverse needs of students, who must navigate their educational journey through this carefully structured time matrix.

The fundamental issue in university course timetabling involves allocating students to classes and determining the specific days and times for each class. At its most basic level, this problem focuses solely on curriculum planning, without incorporating student-specific considerations.

In addressing the university course timetabling problem, our proposed solution approach consists of a formulation of the timetabling problem as a mixed-integer linear program. The project will divide the timetabling problem into two parts: time and room assignment of classes and student sectioning. Our objective is to minimize the total penalty arising from various scheduling conflicts and resource allocation inefficiencies, which is the sum of two weighted penalties: time penalty and room penalty for each class.

In the realm of time and room assignment, the challenge is to allocate classes to specific times and spaces, each bound by its own set of constraints and characteristics. This process must consider a range of variables, from the timing parameters of each class — such as week, day, start time, and duration — to the availability of the rooms themselves. This planning is further complicated by the need to minimize overlaps in schedules, which is crucial for preventing conflicts for students and faculty involved in multiple courses. Parallel to this is the challenge of student sectioning, a task that requires balancing student preferences with the logistical constraints of course offerings and classroom capacities.

2 Problem Description and Formulation

2.1 Problem Description

In this timetabling optimization problem, we need to find the optimal class schedules and room assignments while adhering to multiple constraints. We have room-related constraints, which include the availability of the room at different times of the semester and the capacity of the room. We have class-related constraints, which include the available rooms, maximum class size, class-subpart-configuration-course relationships, and prerequisite class (represented as parent-child relationships in this problem). We also have student-related constraints, which indicate the course selection of each student.

2.2 Decision Variables

In this model, we incorporate three primary decision variables into our framework. The first is the Class-Time Variable ($X_{c,t}$), a binary variable indexed by classes and times. The second is the Class-Room Variable ($Y_{c,r}$), another binary variable, but it is indexed by classes and rooms. The final

one is the Student Sectioning Variable ($Z_{s,c}$), which is also a binary variable, but it is indexed by students and classes. These variables are integral to the decision-making process of the model.

Variable	Value
$X_{c,t}$	1 if class c is scheduled in time t; 0 otherwise
$Y_{c,r}$	1 if class c is scheduled in room r; 0 otherwise
$Z_{s,c}$	1 if student s is assigned to class c; 0 otherwise

Table 1: Variables used in the formulation.

2.3 Objective Functions

Like timetabling in real-life, a class might have multiple options when it comes to time and room assignments. However, some classrooms and time periods are preferred over the others because they might have better availability. To represent this, we introduced the concept of 'penalties' in our objective function to ensure the flexibility of feasible solutions and provide criteria to compare between different solutions. Time and room penalties are the penalties that are given for each possible assignment of a class to a time slot and a room. These penalties reflect the suitability of the assignment for the class, based on the preferences of the students and instructors, and the characteristics of the rooms. For example, a class may have a high penalty if it is assigned to a time slot that is too early or too late, or to a room that is too far away or too small. We aim to minimize the total time and room penalty of all the assignments:

$$\sum_{c,r} Y_{c,r} \cdot \text{room\_penalty} + \sum_{c,t} X_{c,t} \cdot \text{time\_penalty}$$

2.4 Constraints

The constraints in our model are categorized into three distinct groups. The first category, known as the primary constraints, ensures the general feasibility of the timetable. The second category, the student sectioning constraints, guarantees that the assignments meet the enrollment requirements. The third category, the distribution constraints, defines the feasibility between the scheduling of classes. Our base model will only consider the first two categories of constraints, which are essential for the timetabling feasibility. The distribution constraints, on the other hand, are more flexible and complex, and we will leave them for the problem extension section, where we will examine how they impact the quality and feasibility of the timetabling solutions.

2.4.1 Primary Constraints

- Every class c must have one and only one time assignment t.

$$\sum_{c,t} X_{c,t} = 1$$

- Every class c must be assigned to one and only one room r.

$$\sum_{c,r} Y_{c,r} = 1$$

- The capacity of each class c must be greater or equal to the number of students who are assigned to class c.

$$\sum_{s,c} Z_{s,c} <= C(\text{capacity of each class c})$$

- When room r is unavailable during certain time periods t, class c cannot be assigned to room r during time t.

$$X_{c,t} + Y_{c,r} <= 1$$

- The class size must be no more than the capacity of the assigned room.

$$Y_{c,r} * (R(\text{capacity of assigned room r}) – C(\text{capacity of class c})) >= 0$$

2.4.2 Student Sectioning Constraints

Course A			
Configuration A			
Subpart A		Subpart B	
		Class C (parent)	Class C (parent)
Class A1	Class A2	Class B1 (child)	Class B2 (child)

Table 2: Course structure demonstration.

- Every student must attend exactly one class c from each subpart of the selected course's configuration. As shown in Table 2, if a student chooses Course A Configuration A, he must choose one class from Subpart A and one class from Subpart B. In other words, he has to choose one class between Class A1 and Class A2. He also needs to choose one class between Class B1 and Class B2.

$$\sum_{s,c} Z_{s,c} = 1, \text{for c in subpart i, for subpart i in configuration j}$$

- If a class c has a parent class, when the student is assigned to class c, he/she must also be assigned to the parent class. As shown in Table 2, no matter the student chooses Class B1 or Class B2, he must choose Class C because Class C is the parent class of Class B1 and Class B2.

$$Z_{s,c} <= Z_{s,c_{parent}}$$

3 Numerical Implementation and Results

3.1 Numerical Implementation

We are utilizing a scheduling simulation dataset which contains 21 courses, 150 classes, 7 rooms, and 19 students.

The 7 rooms have different capacities and availability (as shown in Table 3). Each time is specified by its starting time within a day and the duration in the number of time slots. There are 288 time slots covering the whole day, with each time slot representing 5 minutes. Days and weeks are specified using binary strings. For example, Room 4 is unavailable on Tuesday ("0100000") of Week 10 ("0000000001000000") starting from 8pm ($\frac{240 \cdot 5}{60}$), lasting for 120 minutes ($24 \cdot 5$).

	Capacity	Unavailable Time
Room 1	1	N/A
Room 2	1	N/A
Room 3	2	days="1000000" start="240" length="24" weeks="0000000001000000" days="0010000" start="240" length="24" weeks="0010000000000000"
Room 4	1	days="0100000" start="240" length="24" weeks="0000000001000000"
Room 5	1	N/A
Room 6	4	days="0100000" start="240" length="24" weeks="0000000001000000"
Room 7	2	days="1000000" start="240" length="24" weeks="0000010000000000" days="1000000" start="240" length="24" weeks="0000000001000000" days="1000000" start="240" length="24" weeks="0000000000010000" days="0100000" start="222" length="12" weeks="0000010000000000" days="0100000" start="222" length="12" weeks="0000000001000000" days="0010000" start="240" length="24" weeks="0010000000000000" days="0001000" start="240" length="24" weeks="0010000000000000"

Table 3: Room Information.

Each of the 150 classes belongs to a course-configuration-subpart relationship. Take Class 42 as an example. Class 42 is part of Course 5 and it belongs to Configuration 5 and Subpart 10, as shown in Table 3. Class 42 has a class size of 1. In this dataset, the class size and room capacity are small, corresponding to the size of small seminars or classes in uncommon study areas (like Jewish Studies). Class 42 also has Class 37 as its parent class. This means students must take Class 37 before taking Class 42. Not all classes have a parent class. For instance, Class 37, 38 and 39 don't have the parent class constraint.

Course 5							
Configuration 5							
Subpart 9			Subpart 10				
n/a	n/a	n/a	parent=37	parent=37	parent=37	parent=37	...
Class 37	Class 38	Class 39	Class 40	Class 41	Class 42	Class 43	...
limit=4	limit=4	limit=4	limit=1	limit=1	limit=1	limit=1	...

Table 4: Course 5 structure demonstration.

Apart from the course-configuration-subpart relationships, the dataset also indicates the room and time options for specific classes. For instance, Class 40 can be assigned to Room 3, Room 4, Room 6, and Room 7. However, different room assignments are related to different penalties. In this case, Room 4 has a lower penalty than Room 3 and Room 7, and Room 6 has the highest penalty. Similar rules apply to Time, as shown in Table 5.

	Room	Time
Class 40	room id="6" penalty="4" room id="7" penalty="1" room id="3" penalty="1" room id="4" penalty="0";	days="0001000" start="90" length="10" weeks="1111111111111111" penalty="64"; days="0000100" start="90" length="10" weeks="1111111111111111" penalty="64"; days="1000000" start="102" length="10" weeks="1111111111111111" penalty="0"; days="0100000" start="102" length="10" weeks="1111111111111111" penalty="0"; ...

Table 5: Class 40 Information.

The rest of the dataset (student's course selection, distribution constraints, etc.) is organized in an intuitive way and thus will not be explained here due to length limitation.

3.1.1 Data Pre-Processing

To enhance the effectiveness of our scheduling model, we have first converted our dataset from XML files to Python objects and then undertaken several pre-processing steps to structure and refine the data:

- Room and Student Identification: We first compiled comprehensive lists of room and student IDs to streamline the referencing process within the model.
  - rooms_id: An array enumerating the IDs of available rooms
  - rooms_id = [1, 2, 3, 4, 5, 6, 7]
  - students_id: An array listing all student IDs
  - students_id = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
- Class Data Structuring: Flatten the structure of courses to get a list of all classes across all subparts and courses. Each class is appended with its subpart ID for later reference. Then a list of class IDs has also been generated.
  - classes = ['id': 1, 'limit': 2, 'room': ['id': 6, 'penalty': 4, 'id': 7, 'penalty': 0, 'id': 3, 'penalty': 0], 'time': ['days': '1010100', 'start': 102, 'length': 10, 'weeks': '1111111111111111', 'penalty': 16, 'days': '...'], 'sbp_id': 1, 'id': 2, ...]
  - classes_id = [1,2,3, ...]
- Enrollment Pairing:
  - Student-Subpart Pairs: For each student and each course they are taking, create a pair of the student ID and the IDs of classes in each subpart of the course. This data structure is helpful for constraints that relate students to specific parts of courses (subparts).
    - student_subparts = [(1, [1]), (1, [142]), (1, [143, 144]), ...]

    This indicates that student with ID 1 is enrolled in a class with ID 1, ID 142, ID 143 and ID 144(which are part of the same course subpart).
  - Student-Classes Pairs: Similar to the student-subpart pairs, but directly linking students to all classes they must attend across all subparts of their courses.
    - student_classes =[(1, [1, 142, 143, 144, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88]), ...]

3.2 Results

We incorporated all constraints into our model and successfully generated the following results:

- Class-time variable $X_{c,t}$: This variable represents the scheduling of classes in specific time slots. Each entry in this variable shows whether a particular class (identified by 'Class') is scheduled in a specific time slot (denoted by 'Time'). For instance, an entry (1, 5, 1) in the Class-Time Variable means that Class 1 is scheduled in the sixth available time slot (as the index starts from 0 in Python).

Class	Time	Value
1	5	1
2	9	1
...	...	...

Table 6: $X_{c,t}$ example results.

- Class-room variable $Y_{c,r}$: This variable indicates the allocation of rooms to classes. Each entry specifies whether a certain class is assigned to a particular room. For example, an entry (1, 2, 1) in the Class-Room Variable implies that the third available room (as the index starts from 0 in Python) is assigned to Class 1.

Class	Room	Value
1	2	1
2	0	1
...	...	...

Table 7: $Y_{c,r}$ example results.

- Student sectioning variable $Z_{s,c}$: This variable deals with the enrollment of students in different classes. An entry in Z indicates whether a student is enrolled in a particular class. For instance, an entry (1, 1, 1) suggests that Student 1 is enrolled in Class 1.

Student	Class	Value
1	1	1
1	142	1
1	144	1
1	82	1
1	88	1
2	...	...

Table 8: $Z_{s,c}$ example results.

To derive the exact timing details Class-time variable $X_{c,t}$, we need to cross-reference Table 5, which contains information on available rooms and time slots for each class. Taking the entry (1, 5, 1) as an example, we refer to the corresponding row in Table 9 for Class 1. Here, the sixth available time slot is characterized by the following attributes:

- Days: '1010100', indicating that the class occurs on Monday, Wednesday, and Friday.
- Start: '162', which translates to a start time of 13:30 hours (calculated as 162 multiplied by 5, then divided by 60 to convert from minutes to hours).
- Length: '10', meaning the class lasts for 50 minutes (10 multiplied by 5).
- Weeks: '1111111111111111', signifying that the class is scheduled for all 16 weeks.
- Penalty: '0', indicating there are no penalties for this time slot.

Similarly, for the Class-room variable $Y_{c,r}$, consider an entry like (1, 2, 1). This indicates that the third available room (as per Python's zero-based indexing) is allocated to Class 1. Referring again to Table 9, the third available room for Class 1 is Room 3, with the following attributes:

- Room ID: '3', identifying the specific room allocated.
- Penalty: '0', indicating that using this room does not incur any penalty.

	Room	Time
Class 1	room id="6" penalty="4" room id="7" penalty="0" room id="3" penalty="0"	days="1010100" start="102" length="10 weeks="1111111111111111" penalty="16"; days="1010100" start="114" length="10" weeks="1111111111111111" penalty="0"; days="1010100" start="126" length="10" weeks="1111111111111111" penalty="0"; days="1010100" start="138" length="10" weeks="1111111111111111" penalty="0"; days="1010100" start="150" length="10" weeks="1111111111111111" penalty="0"; days="1010100" start="162" length="10" weeks="1111111111111111" penalty="0"; days="1010100" start="174" length="10" weeks="1111111111111111" penalty="16"; days="1010100" start="186" length="10" weeks="1111111111111111" penalty="16"
Class 2	...	...

Table 9: Class 1 Information.

To construct a precise timetable for Student 1 in week 1, we first need to determine which classes the student is enrolled in. This information is derived from the Student Sectioning Variable ($Z_{s,c}$). Once the classes are identified, we then refer to the Class-Time Variable ($X_{c,t}$) and Class-Room Variable ($Y_{c,r}$) to ascertain the specific times and rooms allocated for these classes.

	Monday	Tuesday	Wednesday	Thursday	Friday
7:30 - 8:20	Class 88 (Room 6) Class 142 (Room 6) Class 144 (Room 7)	Class 82 (Room 4)	Class 88 (Room 6) Class 142 (Room 6) Class 144 (Room 7)	Class 82 (Room 4)	Class 88 (Room 6) Class 142 (Room 6) Class 144 (Room 7)
13:30 - 14:20	Class 1 (Room 3)		Class 1 (Room 3)		Class 1 (Room 3)

Table 10: Student 1 week 1 timetable.

Upon generating the initial schedule using the decision variables, it was observed that there is a scheduling conflict for Student 1. Specifically, three classes—identified as Class 88, Class 142, and Class 144—are currently assigned to the same time slot. This conflict may be due to the fact that the current formulation of the objective function only includes penalties for room and time conflicts but omits penalties for student scheduling conflicts. This oversight can lead to situations that mirror common dilemmas faced by students in real life, where it is not always possible to enroll in all preferred courses due to overlapping schedules. The primary focus of the model is optimizing the use of rooms and times, which are fixed resources, and it is practically impossible to avoid all scheduling conflicts due to the sheer number of possible course combinations and the limited number of available time slots.

4 Problem Extensions

To address the time conflicts in the base model, we added two additional distribution constraints, SameAttendees and NotOverlap. However, the distribution constraints took huge computational resources and dramatically slowed down the speed of finding the optimal solution. When we added the NotOverlap constraints, our program was automatically interrupted after running for 15 hours due to not having enough computational resources. To address this issue, we further optimized our model through setting limitations on the solver's running time and solution gap (refer to section 4.2).

4.1 Distribution Constraints

4.1.1 SameAttendees

The SameAttendees constraint means that these classes cannot overlap in time because there are students who are studying both of them. For example, if a student is taking two courses that have a SameAttendees constraint, he or she cannot attend both courses if they are scheduled at the same time. The SameAttendees constraint is used to avoid student conflicts and ensure that students can attend all the classes they need. As a result, if a class c_1 is scheduled at time t_1, then another class c_2 cannot be scheduled such that the times overlap.

$$X_{c_1,t_1} + X_{c_2,t_2} <= 1, \text{if } t_1 = t_2$$

$$Z_{s,c_1} + Z_{s,c_2} + X_{c_1,t_1} + X_{c_2,t_2} <= 3, \text{for all class pairs } (c_1,\, c_2)$$

4.1.2 NotOverlap

The NotOverlap constraint says that two classes cannot happen at the same time in the same room.

$$X_{c_1,t_1} + X_{c_2,t_2} + Y_{c_1,r} + Y_{c_2,r} <= 3$$

where c_1, c_2 represent different classes and Room r is a potential location for both classes.

4.1.3 Data Pre-Processing

To satisfy our constraints, we apply some data pre-processing steps before developing the second MIP model.

- Identifying Class Pairs with SameAttendee Constraint: SameAttendee pairs is, for each constraint of type 'SameAttendees', create pairs of class IDs. These pairs are used in the optimization model to ensure that classes with the 'SameAttendees' constraint do not clash in scheduling.
  - SA_pairs = [(1, 31),(1, 32),(1, 37),(1, 38),(1, 39),. . .]

  The tuple (1, 31) means that the same students are enrolled in both Class 1 and Class 31. Therefore, the scheduling should ensure that these classes do not clash in the timetable, allowing the same group of students to attend both without any scheduling conflicts.

4.2 Time and Gap Limits

In optimization problems, apart from getting the optimal solution, the efficiency of achieving the solution is also an important factor to consider. In Gurobi, efficiency can be manipulated through

setting the time and gap limit. Time means the maximum time allowed for the solver to find the local optimal solution and Gap represents the difference between the best-known solution found by the solver and the best possible solution, expressed as a percentage or absolute value. We tried three Time-Gap combinations, [Time=0.5-hour, Gap=0.5], [Time=1-hour, Gap=0.5], and [Time=1-hour, Gap=0.1], and compared the corresponding results.

4.3 Numerical Results

For [Time=0.5-hour, Gap=0.5], we get four feasible solutions, with the minimized objective functions equal to 16,17, 37, and 132. Similarly, for [Time=1-hour, Gap=0.5] and [Time=1-hour, Gap=0.1], we get the same four feasible solutions, with objective functions equal to 16, 17, 37, and 132. As a result, the optimal solution among the feasible solutions is a minimal penalty score of 16.

Following the enhancement of the scheduling model with the SameAttendees, NotOverlap constraints and Time and Gap Limits (Time=1-hour, Gap=0.5), there has been a shift in the timetable for Student 1 during the first week. Although Student 1 remains enrolled in the same classes, the timings and room allocations have been adjusted. Despite these improvements, a degree of class conflict persists, reflecting the inherent challenges of scheduling within a constrained environment.

	Monday	Tuesday	Wednesday	Thursday	Friday
9:30 - 10:20	Class 1 (Room 7) Class 88 (Room 6)		Class 1 (Room 7) Class 88 (Room 6)		Class 1 (Room 7) Class 88 (Room 6)
10:30-11:20 (Mon./Wed./Fri.) 10:30-11:45 (Tue./Thu.)	Class 142 (Room 6) Class 144 (Room 7)	Class 82 (Room 7)	Class 142 (Room 6) Class 144 (Room 7)	Class 82 (Room 7)	Class 142 (Room 6) Class 144 (Room 7)

Table 11: Student 1 week 1 timetable after extension.

In our basic model, we only considered the most essential constraints in timetabling problems. However, real-life situations are much more complicated. Through including SameAttendees constraints, we addressed situations like two classes are taught by the same professor or the same students taking these two classes (thus cannot take place at the same time). Through including NotOverlap constraints, we are dealing with the integrated timetabling problem, considering how the room and time assignment of one class will impact the other instead of treating each class separately. These two additional constraints lead to the reduced class conflicts as shown in the schedule sample above.

While adding additional constraints enable our solution to be more adhered to reality, it also significantly increased the computational resources needed. Our dataset only contains 21 courses, 150 classes, 7 rooms, and 19 students, yet it already took more than 15 hours and still unable to find the optimal solution. Imagine if we want to find the best schedule for a university with over 20,000 students. It will take the solver forever to find the best solution. In the timetabling problems, it is worthwhile to sacrifice some gaps between the best solution and a good enough solution in exchange for the saved time. In this case, the time and room assignment of certain class might not be the most ideal option, yet considering the overall performance of the model, they are the best solution we can get.

5 Recommendations and Conclusions

After conducting comprehensive experiments involving the dataset and our current model, we have identified several pivotal aspects that warrant careful consideration for the advancement of this project. These considerations encompass refining the model's performance, optimizing dataset utilization, and addressing any shortcomings identified during the experimental phase. Additionally, we aim to explore avenues for enhancing scalability, improving computational efficiency, and aligning the project with evolving requirements. This iterative approach ensures a nuanced understanding of the project's dynamics, laying the groundwork for informed decisions and continuous improvement.

5.1 Data Quality and Preprocessing

- Difficulty: The dataset obtained from the website is in XML format, reducing its readability and complicating information extraction while aligning its structure with Python. The current limited test sample may result in suboptimal or even infeasible timetables, as it lacks diverse values typical of real-life scenarios. For instance, a room with a capacity of 1 is an unrealistic constraint that may skew the optimization process.

- Recommendation: We strongly advise future consultants to develop a robust XML parser for swift conversion of XML files, which is our initial dataset format, into Python objects. This preprocessing step transforms the data into a suitable format, enabling efficient handling of the intricate and varied data structures inherent in the timetabling problem.

- Learning: Data preprocessing stands as a crucial phase in guaranteeing the validity and reliability of our timetabling solution. We have dedicated significant time to this step, rigorously experimenting with various datasets to verify data accuracy. This meticulous approach not only ensures the precision of the input dataset but also enhances our ability to optimize the timetabling solution efficiently.

5.2 Model Complexity

- Difficulty: The existing model poses a formidable challenge, featuring over 6000 decision variables and 600,000 rows of constraints. Despite its intricacy, this complexity has translated into extended optimization durations, often yielding only marginal improvements in solutions. In an illustrative instance, our initial model, devoid of time and gap restrictions, ran for over 24 hours before being interrupted due to the computational limitations of our laptops. Also, since we did not set up penalties for class conflicts, our resulting timetable from above still has overlapping classes for one student at one time.

- Recommendation: Future investigations must carefully weigh the trade-off between model complexity and scalability, taking into account the balance between expressiveness and the ability to handle bigger instances, especially when we start building MVP models. Meanwhile, we will introduce penalties for overlapping courses to make sure conflicts for student timetables are minimized.

- Learning: Expanding on the earlier concept, we conducted experiments by adjusting solver parameters to identify the optimal configuration for our problem. The meticulous fine-tuning of parameters, including tolerance levels, presolve alternatives, and solution gaps, not only impacted

solution quality and computation time but also generated viable solutions for our selected problem. Looking ahead, there is potential for further exploration with larger datasets, introducing additional constraints to address diverse scenarios such as travel time between classes or student workload. This approach aims to enhance the model's precision and applicability in real-life situations

5.3 User Interface and Interaction

In our forthcoming implementations, we are committed to delivering a user-friendly interface that prioritizes an optimal user experience, catering to stakeholders like administrators, students, and faculty members. This interface will facilitate the effortless submission of preferences, timeline checks, and necessary adjustments. Through active collaboration with end-users in the design process, we aim to align the interface with their individual demands, thereby enhancing the overall usability of the timetabling system. However, this must be achieved while considering the inherent complexity of the model and addressing evolving demands.

Moreover, in terms of design considerations, we envision additional features. Users will have the capability to explore diverse alternatives, scenarios, and conduct what-if analyses for the timetabling problem. They can adjust parameters, weights, and objectives, observing the real-time impact on solution quality and feasibility. Additionally, users will enjoy the flexibility to validate, edit, or delete their data as needed. The interface will boast a clear and comprehensive visualization of the timetabling solution through various views and formats, such as tables, charts, calendars, and maps. Users can further utilize functionalities like filtering, sorting, searching, and comparing timetabling results for a more intuitive and tailored experience. This comprehensive approach to design aims to offer not only user-friendly interactions but also versatile tools for efficient timetabling management.

5.4 Conclusion

In summary, our current timetabling model satisfactorily meets essential criteria, demonstrating its competence in handling typical scheduling scenarios. However, as we scrutinize its performance, we recognize diverse opportunities for refinement that could significantly enhance its applicability, especially in the context of larger and more complex scheduling situations.

The existing model, while effective, may benefit from adaptations to better accommodate the intricacies and scale of larger scenarios. Potential enhancements encompass optimizing algorithmic efficiency, streamlining data processing, and fortifying the model's robustness to ensure seamless operation under increased workload and complexity.

By strategically addressing these areas, we envision not only improving the model's scalability but also unlocking its potential for handling a broader range of scheduling challenges. This proactive stance toward refining our timetabling model underscores our commitment to continual improvement and adaptability in response to the evolving demands of scheduling in various contexts.