TIMETABLE ARCHITECT

Dr Praveena T, Vineeth Rao, Shreya Prasad

Abstract— Timetable creation in universities is complex, requiring coordination of teacher availability, classroom allocation, and course requirements. Manual timetabling in VTU colleges is labour-intensive and prone to conflicts. This research develops software to automate the process, ensuring conflict-free schedules. The software follows a deterministic algorithm whose output can be tuned to the institution's preferences by tweaking a few weights. The software streamlines scheduling, enhances resource utilization, and provides flexibility for adjustments, improving the efficiency of academic scheduling.

I. Introduction

Creating university timetables is a complex and time-consuming task which requires managing various interconnected elements, including the availability of teachers, subject requirements, and classroom time constraints. At present, the timetables in VTU colleges are generated manually which is a cumbersome process. There is a separate committee created for timetable creation which collects data about teachers and classroom availability from various departments and then formulate the timetable. The primary aim of this research is to design software that automates the timetable creation process, eliminating any conflicts among teachers, subjects, and classrooms. This paper will delve into the challenges of timetable creation, outline the methodology employed in developing the software, and present the outcomes of implementing the software within a university context.

II. Literature Review

Timetable creation for universities can be considered as an NP hard problem. NP-hard problems are computation challenges where verifying a solution is easy but finding the solution itself is very difficult. Various algorithms have been tried to be implemented to solve this problem.

While creating an optimal timetable, various constraints have to be kept in mind. These constraints can be classified as hard constraints and soft constraints [1].

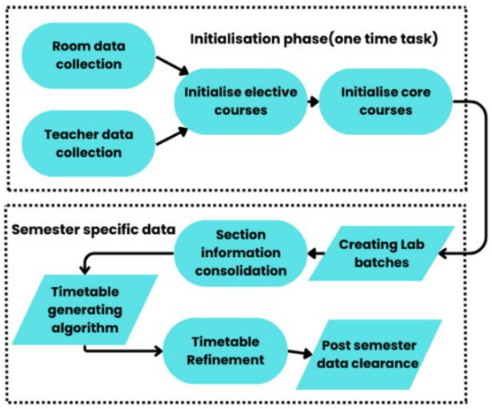
Hard constraints are the constraints that are to be mandatorily satisfied whereas soft constraints are the constraints that are optional and only help in generating better timetables. There are over hundreds of papers which discuss about timetable generators. Multiple algorithms have been explored to come up with the most optimal timetable. The most popular among them are genetic algorithms and Tabu search method. Genetic Algorithms are powerful general purpose optimisation tools which model the principles of evolution [2,3]. They are often capable of finding globally optimal solutions even in the most complex of search spaces. Genetic algorithms work well for flexible problems, but university timetables have many hard constraints like professor availability and room capacities. Finding a valid solution can be time-consuming, and even then, optimal solutions, which are a perfect fit for everyone, might be very difficult to achieve. Also, real-world university timetabling often involves complex negotiations and preferences that are challenging to model effectively with a genetic algorithm.

Another popular algorithm that can be considered for solving this problem is Tabu Search Algorithm [4]. Tabu search is a meta-heuristic procedure for solving optimisation problems. Unlike some stricter optimization algorithms, Tabu search allows revisiting previously explored options under certain conditions. This flexibility proves valuable in university timetabling where adjustments and trade-offs might be necessary to create a workable schedule. However, Tabu search can take a long time to find the best solution, especially for very large universities with many classes and students. The graph colouring algorithm for scheduling a university timetable assigns different colours (time slots) to vertices (courses) of a graph, ensuring that no two adjacent vertices (conflicting courses) share the same colour, thus optimizing the schedule to avoid overlaps [5].

Integer Linear Programming (ILP) is a mathematical optimization technique used to schedule university timetables by assigning classes, instructors, and rooms in a way that satisfies constraints like availability and room capacities while optimizing for objectives such as minimizing schedule conflicts or maximizing resource utilization [6].

III. Overview

The module for the generation of university timetables normally has a structured process of development and mainly consists of two phases: Initialization and Semester-Specific Processing. Initialization is a one-time activity and consists of gathering and storing necessary data, which includes classroom data, comprehensive profiles for all teaching staff and list of courses, both elective and core courses, offered in the university. The assembled data is consolidated in the Semester-Specific Processing Phase which is repeated every semester in order to create comprehensive section information. An algorithm is then executed to generate an initial timetable, which can be refined later to meet specific needs and constraints. It also involves the organization of data into manageable batches in order for it to be processed with efficiency and data stored to be cleared at the end of each academic year, getting ready for the next cycle. Such a workflow from data collection through timetable refining up to data clearance ensures the provision of smooth, flexible, and efficient scheduling of activities that leads to more utilization of resources with minimal conflict.


Initialisation phase(one time task)
Room data collection
Teacher data collection
Initialise elective courses
Initialise core courses
Semester specific data
Section information consolidation
Creating Lab batches
Timetable generating algorithm
Timetable Refinement
Post semester data clearance

The core of the timetable generation module is a deterministic algorithm designed to produce optimized schedules by assigning specific weights to various factors. This approach allows the software to prioritize different scheduling elements such as teacher availability, classroom capacity, and course requirements. By adjusting these weights, the algorithm can be fine-tuned to meet the specific preferences and constraints of the institution. The deterministic nature of the algorithm ensures that given the same inputs and weights, the output will always be consistent, providing reliable and predictable outcomes. This method not only automates the complex task of timetable creation but also offers the flexibility to adapt to changing needs and thereby enhancing the efficiency of timetable scheduling.

IV. Implementation

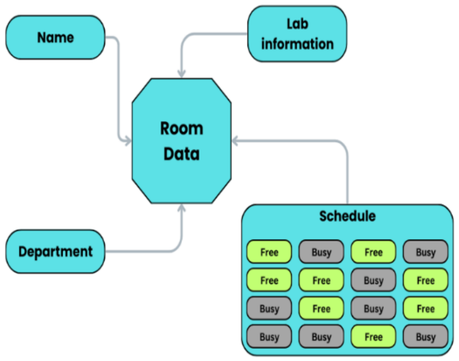
A. Inputs

Taking inputs is the most crucial part of timetable creation. Inputs are crucial for the university timetable software because they provide the basic data needed to create accurate and effective schedules. Without comprehensive data, the software cannot effectively assign faculty, rooms, or balance the needs of students and instructors. To ensure a structured approach, inputs are collected in stages:

1. Classrooms:

•Classroom Name:
Each classroom is identified by a unique name or identifier, such as CS101, to distinguish it from other classrooms

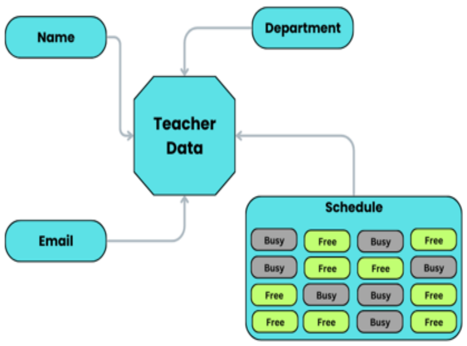
•Current Occupancy Details:
Information on the times when the classroom is already in use is gathered. This helps to integrate the existing schedule into the new timetable, preventing any scheduling conflicts and ensuring smooth operations.



2. Teachers

•Teacher Name:
The name of each teacher is collected, along with additional details such as their department and college email ID for unique identification.

•Availability Schedule:
Detailed availability schedules for each teacher are gathered, indicating when they are available to teach and when they have other commitments. This helps avoid conflicts with their existing obligations and ensures that new schedules fit seamlessly with their current commitments.



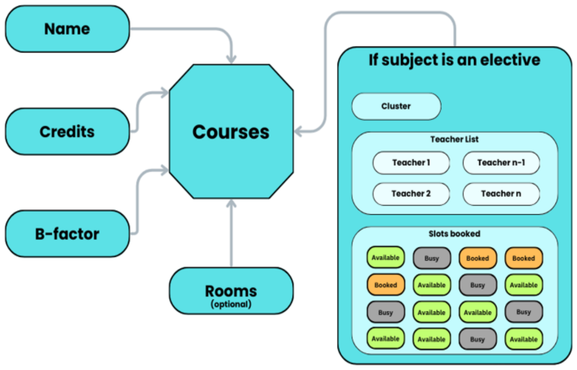
3. Courses
   - Course Name:
     The name or code of each course is collected to identify the course being scheduled.

   - Credits:
     The number of credits associated with each course is recorded, indicating its workload and duration.

   - Elective Status:
     Whether the subject is an elective or required course is specified, helping to meet program requirements and student preferences. This will be discussed further in the paper

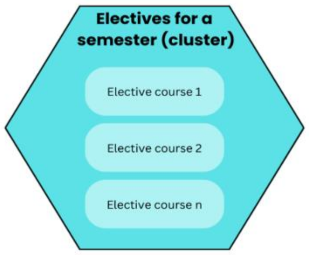
   - Preferred Room Allocation:
     If there are specific rooms preferred or required for teaching a course, this information is noted to optimize scheduling and room allocation.

   - B-Factor:
     The "B-factor," which evaluates the difficulty level of a course. This metric ensures a balanced distribution of demanding courses throughout the student timetable, preventing an excessive concentration of challenging classes on any single day.



It is important to note that this method of input collection does not apply to lab courses, which will be discussed later in the paper

In university scheduling, a cluster is a group of related elective courses. Imagine it like a logical group of categories of electives. Students pick one subject for each category to fulfill their elective requirement. The software makes scheduling electives easier by finding a shared free time slot for all the teachers for each category within a cluster.
The software doesn't pick the timeslot itself. Instead, it presents all the available common time slots to the user. They can then choose the timeslot that best suits everyone's needs, considering factors like student preferences or avoiding scheduling too many electives at the same time during the day.
For example, Programming Language Course (PLC) could be an elective course under 2ndSem Cluster. PLC may contain various elective subjects like C++, python, and java under it. Since different students may choose different subjects, the software finds and displays common free time slots for all the teachers handling the chosen subjects for that particular course.



B. Labs

After the inputs of teachers and classrooms are taken, laboratory subjects can be created.

Since typically laboratory rooms are fewer than normal classrooms and require more than one teacher to conduct a session, we prioritize allocating laboratory subjects first for a better and more flexible schedule. We achieve this by bundling all the laboratory courses and their inputs for a class into a "batch." When generating the section, we add this batch of courses to the section.

To create a lab, the following input from the user is required:

1. Batch name: The name of the batch being created. For example, "4(C1/C2/C3)" can be a batch name for the lab subjects for students in the 4th semester, section C, if three subdivisions exist in that batch.

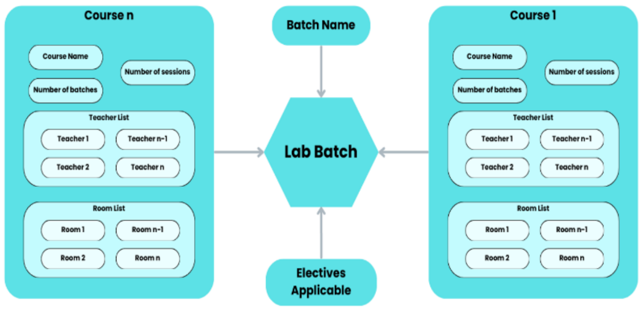
2. Course information:

   - Lab course name: The name of the course to be added to the batch. One batch can have multiple courses.
   - Number of batches: Normally, one section is divided into multiple (student) batches (Not to be confused with the batch we are making) with each batch having its own room and teachers.

- Number of sessions: The number of sessions of this particular course to be conducted across a week.
- Teachers handling the course: A list of teachers assigned to conduct the lab subjects. The number of teachers entered should be equal to or greater than the number of teachers required per batch.
- Number of teachers per batch: The number of teachers conducting that particular course.
- Rooms for the course: The rooms that can accommodate the course batch. The number of rooms entered should be equal to or greater than the number of batches.

Example:

- Course name: IOT
- Number of batches: 3
- Number of sessions: 1
- Teachers handling the course: AA, BB, CC, DD, EE
- Number of teachers per batch: 1
- Rooms for course: Lab1, Lab2, Lab3, Lab4



3. Elective information: All electives applicable to that particular section. For example, ABC elective.

4. Blocked timings: Timings where courses should not be allotted. For example, all Saturday periods and all periods when math courses are scheduled.

After taking all the necessary inputs, the automatic allotment can take place.

First Step of Allotment is to block all slots where the labs cannot be allotted. This includes all the electives that have been allotted and also the blocks they have given in the inputs.

Then we need to choose the best possible combination of teachers and classes. Next steps are to make all combinations of teachers and classes, then see all the possibilities, and select the best one.

After getting all the possible combinations of teachers and rooms, we have to find the feasible slots to allot the teachers, which we get by following all the hard constraints.

Following are the hard constraints:

1) Teacher cannot be in 2 classes simultaneously.
2) 1 Room cannot accommodate more than 1 batch.
3) Each lab allotment has to be 2 periods long.
4) Lab slots cannot have break times within it.
5) There can be only one lab per day.

By iterating through each teacher and rooms in every combination we created earlier and by obeying all the hard constraints we can filter out all the valid combinations from the previously created list of combinations.

Next step is to choose the best combination possible. We do this by ranking all the combinations based on how much they obey soft constraints.

1) The teachers conducting the lab should be free before or after the session.
2) The teachers chosen should be the ones who are the most free compared to other teachers

The weightages given to each soft constraint can be changed manually, to further change the allocation based on the institution's preference.

Its described as:

$$\text{Intersection (p , d)} = \begin{cases} 1: \mathit{if\ period\ is\ invalid} \\ 0: \mathit{if\ all\ hard\ constrainsts} \\ \qquad \mathit{are\ obeyed} \\ 1: \mathit{otherwise} \end{cases}$$

$$\text{Status (p , d , t)} = \begin{cases} 1: \mathit{if\ period\ is\ invalid} \\ 0: \mathit{if\ teacher\ is\ busy\ in\ the} \\ \qquad \mathit{corresponding\ time} \\ 1: \mathit{otherwise} \end{cases}$$

$$\text{Value (p, d, t)} = \sum_{i=-1}^{1} Status(p+i,d,t) \times \ w_i \times Intersection(p+i,d,t)$$

$$\text{Score } (t_1, t_2, \ldots, t_n, r_1, r_2, \ldots, r_n) = \sum_{t_i=t}^{t_n} \left(\sum_{d=0}^{d=\max} \left(\sum_{p=0}^{p=max} Value(\,p, d, t_i\,) + w_k \times \text{Status (p , d , t)} \right) \right)$$

Where p: period, d: day, $t_1, t_2, \ldots, t_n$: Teachers of the combination. $r_1, r_2, \ldots, r_n$: Rooms of the combination.

The score of each combination is used for ranking.

After ranking all the possible combinations, the best possible slots (using the sum of Value of all teachers at different periods to decide) of the best combination for each subject are selected and courses are allotted at that particular timeslot.

FLEXIBLITY:

After the courses are allotted, we provide the user with the flexibility to change the allotted slot to any other slot, since there might be other special conditions which cannot be generalized for all the courses/teachers/rooms. We do this by conveying to the user all possible slots where all hard constraints are obeyed, and hint them the slots where soft constraints are obeyed more than other slots. For example, the slots where hard constrains are obeyed might be shown in green, while others are shown in gray, and different shades of green are shown based on how much the soft constraints are obeyed.

After all the above are done, every lab can be saved as its batch name, to be accessed later when allocating sections.

Note:

Some institutions allot different courses for each sub-batch in a lab session. This can also be resolved by giving a different set of teachers for each sub-batch and different set of rooms for each sub-batch.

For example, if for a batch with 3 sub-batches, 2 different courses, IOT and DAA are to be allotted as IOT/IOT/DAA and DAA/DAA/IOT, then one course can be named as IOT/IOT/DAA and 2 teachers of IOT entered, 1 teacher of DAA entered, same with the rooms and repeated similarly for the other course as well, the allocation will be as expected from the user.

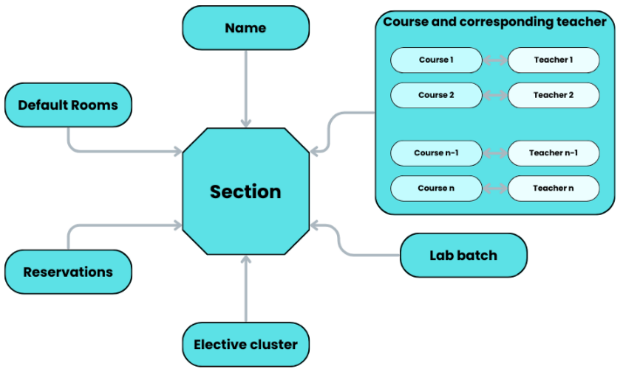
C. *Sections*

After the formation of lab batches, we can move on to create sections.

Section requires the following inputs:

- Electives: Elective courses assigned for the section
- Lab subjects: Batches of lab made earlier for this particular section
- Courses and teachers: Courses assigned for and teachers teaching the courses
- Rooms where courses can be conducted
- Blocked timeslots: Timeslots where no courses can be allotted.

After these inputs are taken, we run the timetable allotting algorithm.



First step again like the above is blocking the slots where we cannot allot any periods. This includes electives, labs, and blocked timeslots.
Then the optimization algorithm comes into the picture. We iterate through each of the given input courses in the order given by the user, mark the feasible regions where all the hard constrains of the course is obeyed, and score the marked slots based on how many soft constrains they obey. Then the best scores are chosen across different days, obeying on the number of credits for that subject.
Hard constrains are similar to that of labs:

1) Teacher cannot be in 2 classes simultaneously.
2) 1 Room cannot accommodate more than 1 section.

Soft constrains being:

1) Teachers needs to be allotted when they are relatively free
2) Hard subjects should be spread evenly across the week
3) All subjects should be conducted in a single room across the week
4) Classes shouldn't have gaps between them

The algorithm starts by identifying time slots where both the teacher and the classrooms are available. This involves checking the free periods of the teachers who are conducting the course and comparing them with the available time slots in the default room timetable or any specific room of the subject.

Once potential time slots are identified, the algorithm evaluates and scores these slots based on how well they meet various soft constraints.

1) Teacher point of view.
   A weightage is given to the teacher being free before the timeslot and after the timeslot. If the teacher is free, the weighted score is added to the timeslot.
2) Student point of view.
   All hard subjects are spread evenly across the week. Each day is assigned a Bfactor, which is

the sum of Bfactors of all the courses allotted on that particular day. This net Bfactor of the day is allotted a weight which is then subtracted from the score of the timeslot. This ensures that bfactor isn't very high any single day.

Classes are preferred to be conducted in one single room across the week to avoid confusion and make it convenient for both teachers and students. All courses are also preferred to be allotted early on the day without having many gaps in between the 2 courses.

Value of each slot will be given by:

$$\text{Intersection (p , d)} = \begin{cases} 1: \mathit{if\ period\ is\ invalid} \\ 0: \mathit{if\ all\ hard\ constrainsts} \\ \qquad \mathit{are\ obeyed} \\ 1: \mathit{otherwise} \end{cases}$$

$$\text{Status (p , d , t)} = \begin{cases} 1: \mathit{if\ period\ is\ invalid} \\ 0: \mathit{if\ teacher\ is\ busy\ in\ the} \\ \quad \mathit{corresponding\ time} \\ 1: \mathit{otherwise} \end{cases}$$

$$\text{Bfactor (d)}= \sum_{p=0}^{p=max} b_p$$

$$\text{Value (p, d, t)} = \sum_{i=-1}^{1} Status(p+i,d,t) \times w_i \times Intersection(p+i,d,t) - w_0 \times Bfactor - w_t \times period$$

Time slots are ranked based on their final scores. Higher scores indicate better adherence to both soft and hard constraints. The best time slots are chosen based on their rankings, aiming to fulfil the credit requirements for the course. Sometimes if the credit requirements are not fulfilled because of the hard constrains, the algorithm checks for other courses which can be moved and make free a timeslot for the course with credit deficit.

Flexibility:

Like the Labs, even here the user is given the choice to edit the allotted timetable. The user is conveyed the eligible slots which obey hard constrains based and how preferrable the eligible slots are based on the scoring system we discussed above.

V. Discussion

VI. Conclusion

In conclusion, the timetable creation software for universities changes the landscape for academic scheduling as it facilitates the allocation of classes, rooms, and teachers. It not only streamlines what has been considered a very complex process but also optimizes resource utilization, minimizes conflicts in schedules, and further adapts to the ever-changing requirements of an academic institution.
Moreover, the software enhances flexibility for both teachers and students. The timetable is scheduled in such a way that it avoids simultaneous allotment of classes for teachers so as to not over burden them. For students, the software ensures balanced course offerings and minimizes scheduling conflicts, thereby facilitating smoother academic progression.
The seamless integration of various constraints and preferences enhances operational effectiveness. At the same time, it builds flexibility in planning that will most probably have a well-organized and more productive educational environment.

References

[1] H. Alghamdi, T. Alsubait, H. Alhakami, and A. Baz, "A Review of Optimization Algorithms for University Timetable Scheduling", Eng. Technol. Appl. Sci. Res., vol. 10, no. 6, pp. 6410–6417, Dec. 2020.

[2] Davis, Lawrence. "Handbook Of Genetic Algorithms." (1990).

[3] Burke, Edmund & Elliman, David & Weare, Rupert. (1994). A Genetic Algorithm Based University Timetabling System. 1.

[4] Islam, Tanzila & Perves, Mohammad & Shahriar, Zunayed & Hasan, Monirul. (2016). University Timetable Generator Using Tabu Search. Journal of Computer and Communications. 4. 28-37. 10.4236/jcc.2016.416003.

[5] Sabar, N. R., Ayob, M., Qu, R., & Kendall, G. (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence, 37, 1-11.

[6] Oladokun, VO and SO Badmus. 2008. "An Integer Linear Programming Model of a University Course timetabling Problem". Pacific Journal of Science and Technology 9(2)426- 431.

[7] Soyemi, Jumoke & John Lekan, Akinode & Oloruntoba, Abiodun. (2017). Automated Lecture Time-tabling System for Tertiary Institutions. International Journal of Applied Information Systems (IJAIS). Foundation of Computer Science FCS, New York, USA. 12. 21-27. 10.5120/ijais2017451700.

[8] Oude Vrielink, Rudy & Jansen, E. & Hans, Erwin & Hillegersberg, Jos. (2019). Practices in timetabling in higher education institutions: a systematic review. Annals of Operations Research. 275. 10.1007/s10479-017-2688-8.

[9] Aziz, N. L. A., & Aizam, N. A. H. (2018, September). A brief review on the features of university course timetabling problem. In AIP Conference Proceedings (Vol. 2016, No. 1). AIP Publishing.