WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SmartSlots: An Automated Timetable Generator for College Scheduling

SASITHARAN .K[1], RANJITH .S[2], SANJAI KRISHNAN .K[3], SINTHANAIVALAVAN .S[4], PAVINKUMAR .R[5], Mr. E. SUBRAMANIAN[6]

[1,2,3,4,5] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.

[6] Project Guide, Sri Shakthi Institute of Engineering and Technology, India.

ABSTRACT :

Preparing a college timetable is a complex administrative task involving academic sections, subjects, faculty members, classrooms, weekly teaching-hour requirements, and fixed period schedules. Manual timetable preparation can lead to overlapping assignments, inefficient resource utilization, and repeated revisions. This paper presents a web-based College AI Timetable Generator that automates timetable construction using a constraint-based scheduling approach. The application manages departments, semesters, sections, subjects, faculty members, classrooms, faculty availability, and combined classes. Its scheduling engine uses an occupancy tracker, heuristic ordering, and a multi-stage allocation process to assign classes while considering resource availability and scheduling constraints. A separate conflict-detection module identifies faculty overlaps, classroom double-booking, section clashes, availability violations, and room suitability issues. The system is developed using React and TypeScript for the frontend and Node.js, Express, and Sequelize for the backend, with support for relational database configurations. The application also provides timetable management, manual editing, generation history, and export functionality. The proposed system aims to reduce manual scheduling effort and improve the consistency of college timetable preparation. Further experimental evaluation is required to measure generation time, scheduling completeness, and conflict reduction.

Keywords: College Timetable, Artificial Intelligence, Constraint-Based Scheduling, Heuristic Algorithm, Conflict Detection, Resource Allocation, Web Application.

I. INTRODUCTION

Timetable preparation is an essential administrative activity in educational institutions. Colleges must coordinate subjects, faculty members, classrooms, student sections, laboratory sessions, and academic schedules while ensuring that all academic requirements are satisfied. Preparing timetables manually becomes increasingly difficult as the number of departments, faculty members, subjects, and classrooms increases.

Traditional timetable preparation requires repeated adjustments to avoid faculty clashes, classroom conflicts, and overlapping classes. Laboratory sessions introduce additional complexity because they often require consecutive periods. Combined classes, faculty availability restrictions, and different academic sc hedules further increase the difficulty of the process.

The College AI Timetable Generator is designed to automate timetable preparation through a web-based application. It uses a constraint-based scheduling algorithm to allocate subjects, faculty members, and rooms to suitable time slots. The system also includes a conflict-detection mechanism to identify invalid assignments and support timetable validation.

The main objective of this project is to simplify college timetable preparation, reduce manual effort, improve resource utilization, and provide a centralized platform for managing academic schedules.

The major objectives are:

1. To automate the generation of college timetables.
2. To reduce conflicts involving faculty members, classrooms, and student sections.
3. To allocate laboratory sessions across suitable consecutive periods.
4. To manage academic resources through a centralized application.
5. To validate generated timetables and identify scheduling problems.
6. To provide timetable viewing, editing, and export facilities.

II. LITERATURE SURVEY

1. Educational timetabling has been widely studied as a combinatorial optimization problem. It involves assigning academic activities to time slots and resources while satisfying various constraints. Constraint satisfaction techniques are commonly used to represent these requirements and identify valid scheduling arrangements.

2. Traditional manual scheduling depends heavily on the experience of timetable coordinators. Although this approach can produce workable schedules for smaller institutions, it becomes time-consuming when many academic constraints must be considered simultaneously. Changes in faculty availability, room allocation, or subject requirements can require extensive timetable revisions.
3. Constraint-based scheduling represents timetable requirements as rules that must be satisfied. Hard constraints include preventing faculty members from teaching two classes simultaneously, avoiding classroom double-booking, and preventing overlapping classes for the same section. Soft constraints represent preferences such as balanced faculty workloads and convenient subject distribution.
4. Heuristic algorithms improve scheduling efficiency by prioritizing activities that have fewer available options. For example, laboratory sessions requiring consecutive periods can be scheduled before theory classes because they have more restrictive placement requirements.
5. Modern web technologies also support the development of centralized timetable management systems. A browser-based interface allows administrators to configure academic information, generate schedules, review conflicts, and export timetables.
6. The proposed project combines constraint-based scheduling, heuristic ordering, conflict detection, and web-based administration into a single college timetable management application.

III. PROPOSED METHODOLOGY

A. System Architecture

The proposed system follows a client-server architecture consisting of a frontend, backend, database, scheduling engine, and conflict-detection module.
The frontend provides the user interface for managing academic resources and viewing timetables. The backend processes requests, applies scheduling logic, and manages database operations. The database stores academic information and generated timetable entries.
The scheduling engine generates timetable assignments using resource-availability checks and heuristic ordering. The conflict-detection module independently examines assignments to identify scheduling violations.
The major components are:

1. Frontend Module: Provides pages for authentication, dashboard access, academic resource management, and timetable visualization.
2. Backend Module: Processes application requests and coordinates scheduling operations.
3. Database Module: Stores departments, faculty members, subjects, rooms, sections, and timetable information.
4. Timetable Generation Module: Allocates academic activities to suitable time slots.
5. Conflict Detection Module: Identifies overlapping assignments and resource conflicts.
6. Export Module: Supports timetable export into formats such as PDF and CSV.

B. Academic Data Management

The system maintains the academic information required for timetable generation. This includes departments, academic years, semesters, sections, subjects, faculty members, classrooms, faculty availability, and faculty-subject assignments.
Each subject is associated with the appropriate academic section and eligible faculty members. Classroom information helps the system select suitable rooms for theory and laboratory sessions.
Faculty availability information is used to prevent assignments during unavailable periods. Subject-hour requirements determine the number of sessions that should be allocated during the academic week.
This centralized data model provides the foundation for automated timetable generation.

C. Period and Break Configuration

The timetable generator uses a bell-schedule service to determine the start and end times of academic periods. The implementation supports two period patterns for different semester groups.
Each pattern defines period durations, breaks, and lunch intervals. The scheduling system uses these time intervals to determine whether classes overlap. Laboratory sessions require special handling because they may occupy multiple consecutive periods. The system checks suitable laboratory starting periods to avoid scheduling sessions across configured breaks or lunch intervals.
This approach supports different academic schedules while maintaining consistent time-based validation.

D. Constraint-Based Scheduling Algorithm

The proposed timetable generator treats academic scheduling as a constraint-based allocation problem. Each class must be assigned a suitable day, period, faculty member, and classroom.
The algorithm uses an occupancy tracker to maintain information about section assignments, faculty time intervals, room time intervals, faculty workloads, and scheduled subject hours.
Before placing a class, the scheduling engine checks whether the required resources are available. A valid assignment must satisfy the applicable constraints.
The main constraints include:

- Faculty constraint: A faculty member cannot teach overlapping classes.
- Room constraint: A classroom cannot accommodate multiple simultaneous classes.

- Section constraint: A section cannot attend two classes at the same time.
- Availability constraint: Faculty members must be available during their assigned periods.
- Laboratory constraint: Practical sessions must fit within suitable consecutive periods.
- Subject-hour constraint: The schedule should satisfy the required weekly teaching hours.

The generator uses heuristic ordering to prioritize classes with more restrictive scheduling requirements. Longer combined classes and laboratory sessions are handled before regular classes.

E. Multi-Stage Timetable Generation

The scheduling algorithm follows a multi-stage process:
Stage 1: Allocation of Long Combined Classes
Combined classes that require longer time blocks are allocated first because they have fewer suitable scheduling options.
Stage 2: Laboratory and Practical Scheduling
Laboratory sessions are assigned to suitable rooms and consecutive periods while considering faculty availability and break intervals.
Stage 3: Regular Combined Classes
Combined classes with less restrictive time requirements are allocated to suitable slots.
Stage 4: Language Subject Allocation
Language subjects are scheduled according to their academic requirements and resource availability.
Stage 5: Regular and Core Subject Allocation
Remaining theory and core subjects are assigned to available periods.
Stage 6: Remaining Subject-Hour Allocation
The generator attempts to schedule subject hours that remain unfulfilled after the earlier stages.
When a class cannot be placed directly, the implementation includes limited relocation-based repair. It attempts to move selected existing classes to alternative slots to make room for a more restrictive assignment.
This process improves scheduling flexibility, although it does not guarantee that every feasible timetable will be discovered.

F. Conflict Detection

The conflict-detection module validates timetable entries independently of the generation process.
It examines time intervals and resource assignments to identify faculty overlaps, classroom conflicts, section clashes, faculty availability violations, and room suitability problems.
The module classifies detected issues according to severity, including errors and warnings. It also provides contextual information about the affected timetable entries.
This separation between generation and validation helps the application identify problems in both automatically generated and manually modified timetables.

G. Timetable Management

After generation, timetable entries are stored in the database and made available through the application interface.
Users can view generated schedules, retrieve previous generations, edit existing entries, and manage individual timetable assignments. The backend also provides operations for creating and deleting entries.
Validation can be performed when entries are modified to identify scheduling conflicts before accepting changes.
The application includes export services for producing timetable documents in PDF and CSV formats, with ZIP packaging for grouped exports.

IV. SYSTEM IMPLEMENTATION

A. Frontend Implementation

The frontend is developed using React, TypeScript, Vite, Tailwind CSS, and React Router. Axios is used for communication with backend APIs.
The interface includes login, dashboard, timetable viewing, coordinator workflow, department management, faculty management, subject management, section management, classroom management, combined-class configuration, and academic configuration pages.
The frontend separates individual pages and reusable components to improve maintainability and support future development.

B. Backend Implementation

The backend is developed using Node.js, Express, and TypeScript. It exposes API endpoints that handle authentication, academic resource management, timetable generation, timetable retrieval, editing, and export operations.
The backend separates route handling from specialized services. The timetable generator handles scheduling, the conflict detector validates assignments, the bell-schedule service calculates period intervals, and the export service produces downloadable timetable files.
This modular design improves code organization and allows individual components to be tested independently.

C. Database Implementation

The application uses Sequelize as its object-relational mapping framework. The database models represent academic entities and their relationships.
The database stores information about departments, faculty members, subjects, classrooms, sections, timetable generations, and timetable entries.
The supplied project configuration supports SQLite and MySQL database options. The database stores generated schedules and allows timetable information to be retrieved and modified through backend operations.

D. Timetable Generation Workflow

The generation process begins when an authorized user submits a timetable-generation request through the frontend.
The backend retrieves the relevant academic information and initializes the scheduling engine. The engine allocates classes according to the configured scheduling stages and resource constraints.
Generated assignments are stored in the database. The conflict-detection module can then examine the assignments and report scheduling issues.
The frontend retrieves the resulting timetable and displays it to the user. Users can review the schedule, make permitted modifications, and export the timetable.

V. ADVANTAGES OF THE PROPOSED SYSTEM

The proposed College AI Timetable Generator provides the following advantages:

1. Automation: Reduces the need for repetitive manual timetable preparation.
2. Conflict Detection: Identifies overlapping faculty assignments, room conflicts, and section clashes.
3. Efficient Resource Allocation: Considers faculty and classroom availability during scheduling.
4. Laboratory Support: Handles consecutive-period requirements for laboratory and practical sessions.
5. Combined-Class Scheduling: Supports sessions shared by multiple academic sections.
6. Centralized Administration: Maintains academic resources and timetable information within a single application.
7. Timetable Editing: Provides facilities to manage generated and manually created entries.
8. Export Functionality: Supports timetable exports in PDF and CSV formats.
9. Modular Architecture: Separates the frontend, backend, scheduling logic, database models, and validation services.

VI. RESULTS AND DISCUSSION

The implemented system provides a web-based environment for managing academic resources and generating college timetables using a multi-stage scheduling algorithm.
The scheduling engine uses heuristic ordering, resource-occupancy tracking, and limited relocation-based repair to allocate academic activities. The separate conflict-detection module checks assignments for faculty overlaps, room conflicts, section clashes, availability violations, and room suitability issues.
The application also supports timetable retrieval, manual entry management, generation history, and export functionality.
However, quantitative performance results must be established through systematic testing. The supplied project source does not establish measured timetable-generation time, conflict-reduction percentages, scheduling-completion rates, or comparative performance against other algorithms. Therefore, numerical performance improvements cannot be claimed without further experiments.

A. Functional Testing

The following tests should be conducted to evaluate the application:

Test Case	Expected Outcome
Generate a timetable for a configured semester	Timetable entries are generated and stored
Assign one faculty member to overlapping classes	Faculty conflict is detected
Assign one classroom to overlapping classes	Room conflict is detected
Assign two classes to the same section and period	Section conflict is detected
Schedule a faculty member during unavailable periods	Availability violation is detected
Schedule a laboratory across a break	Invalid placement is prevented or reported
Modify an entry to introduce a conflict	Validation identifies the conflict
Export a generated timetable	The requested export file is produced

The actual result of each test should be recorded after executing the application.

B. Limitations

The current scheduling approach relies on heuristic ordering and limited relocation-based repair. Consequently, the algorithm may fail to find a valid timetable even when a feasible arrangement exists.

The final schedule may also contain unfulfilled subject-hour requirements if suitable slots cannot be found. These conditions must be reported clearly rather than treated as a completely successful generation.

The system's effectiveness depends on the accuracy of academic data, faculty-subject mappings, room availability, period configuration, and weekly teaching-hour requirements.

Further evaluation is required to measure scheduling completeness, execution time, resource utilization, and the number and severity of conflicts.

VII. CONCLUSION

This paper presents a web-based College AI Timetable Generator designed to automate academic timetable preparation. The system combines centralized academic data management, constraint-based scheduling, heuristic ordering, occupancy tracking, conflict detection, and timetable administration.

The scheduling engine uses a multi-stage allocation process to handle combined classes, laboratory sessions, language subjects, and regular academic subjects. A separate conflict-detection module checks timetable entries for faculty overlaps, classroom conflicts, section clashes, availability violations, and room suitability issues.

The application is implemented using React and TypeScript for the frontend and Node.js, Express, and Sequelize for the backend. It also provides timetable editing, generation history, and export functionality.

The proposed system provides a foundation for reducing manual scheduling effort and improving the organization of college timetables. However, its reliability and effectiveness must be established through comprehensive functional testing and quantitative performance evaluation. Future enhancements can focus on improving scheduling completeness, optimizing resource utilization, and providing stronger validation guarantees.

VIII. FUTURE ENHANCEMENTS

Future development can improve the system in several ways.

First, a more advanced constraint optimizer or deeper backtracking algorithm could explore additional scheduling alternatives and improve the likelihood of finding feasible timetables.

Second, an improved timetable-quality scoring mechanism could evaluate faculty workload balance, room utilization, subject distribution, and unscheduled teaching hours.

Third, automated regression tests could validate the scheduling engine against a range of academic configurations, including large departments, multiple laboratory sessions, combined classes, and different faculty availability patterns.

Fourth, the application could provide more detailed explanations when a timetable cannot be completed, identifying the specific subjects, resources, or constraints responsible for the failure.

Finally, deployment monitoring, timetable comparison, configurable institutional rules, and enhanced reporting could improve the system's suitability for real-world college administration.

REFERENCES :

1. S. E. Daskalaki, T. Birbas, and E. Housos, “An Integer Programming Formulation for a Case Study in University Timetabling,” European Journal of Operational Research, vol. 153, no. 1, pp. 117–135, 2004.
2. J. H. Holland, Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: University of Michigan Press, 1975.
3. React, “React Documentation.” Available: React Official Documentation
4. Express.js, “Express Documentation.” Available: Express Official Documentation
5. Sequelize, “Sequelize Documentation.” Available: Sequelize Official Documentation
6. Vite, “Vite Documentation.” Available: Vite Official Documentation
7. Tailwind Labs, “Tailwind CSS Documentation.” Available: Tailwind CSS Official Documentation
8. PDFKit, “PDFKit Documentation.” Available: PDFKit Official Documentation