Murali Parameswaran - ISR-03_Report_Examination Timetable Generation[1]

by Murali Parameswaran

Submission date: 11-Jan-2025 02:42PM (UTC+0530)
Submission ID: 2562459194
File name: ISR-03_Report_Examination_Timetable_Generation_1.docx (10.16M)
Word count: 6820
Character count: 47358

Examination Timetable Generation

A PROJECT REPORT

Submitted by,

Susheeth G	20211ISR0036
Ritish N	20211ISR0047
Vidyashree BN	20211ISR0039
Mithali S Anand	20211ISR0082
Tejashwini BA	20211ISR0040

Under the guidance of,

Dr. Murali Parameswaran

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION SCIENCE AND ENGINEERING

[ARTIFICIAL INTELLIGENCE AND ROBOTICS]

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

BENGALURU

DECEMBER 2024

ABSTRACT

The Automated Examination Scheduling System is an innovative solution that utilizes a genetic algorithm to efficiently manage the complex task of creating exam timetables. Designed for educational institutions, the system addresses key challenges such as avoiding exam clashes for students, adhering to room capacity constraints, balancing invigilator assignments, and accommodating diverse exam durations and types, including midterms and end-term examinations.

The scheduling process begins with the analysis of data from user-provided CSV files, detailing student course enrollments, room capacities, invigilator availability, and course schedules. The genetic algorithm employs evolutionary techniques such as population generation, fitness evaluation, crossover, and mutation to iteratively optimize the timetable. Fitness constraints ensure that student conflicts, room overbooking, and consecutive invigilation for the same individual are avoided, producing schedules that are both feasible and fair.

The system is enhanced with a web-based interface developed using Flask, offering a user-friendly platform for administrators. Key features include secure login, data uploads, exam configuration, and automated generation of PDF reports. These reports provide comprehensive documentation, including daily exam schedules, invigilation assignments, and room allocations for students.

By automating the exam scheduling process, this system significantly reduces administrative workload, eliminates manual errors, and ensures fairness and efficiency. Its adaptability makes it suitable for various institutional needs, demonstrating the practical application of genetic algorithms in solving complex optimization problems.

CHAPTER-1

INTRODUCTION

1.1 Introduction

Exam scheduling is a complex and crucial task for educational institutions that involves balancing multiple constraints while ensuring a seamless examination process. This task requires allocating rooms, assigning invigilators, and creating a timetable that accommodates diverse student groups and avoids clashes in exam timings. Traditionally, these schedules have been created manually, which can be tedious, time-consuming, and prone to errors, particularly when dealing with large datasets and intricate constraints. Such inefficiencies can lead to operational disruptions, student dissatisfaction, and logistical challenges.

To address these issues, this project proposes an Automated Examination Scheduling System powered by a genetic algorithm. The genetic algorithm is a powerful optimization technique inspired by the principles of natural selection, where candidate solutions evolve over successive generations to achieve optimal results. This method is particularly suited for handling the multi-dimensional constraints inherent in exam scheduling, such as:

- Avoiding clashes in exam timings for students enrolled in multiple courses.
- Ensuring room capacity limits are respected.
- Fairly distributing invigilation duties among available staff.
- Supporting different exam formats, such as midterms and end-term exams, with varying durations.

The system processes input data from user-provided CSV files, which include detailed information about students, courses, rooms, and invigilators. Using this data, the genetic algorithm generates a pool of potential schedules and iteratively refines them based on fitness criteria. Fitness measures include compliance with scheduling constraints, room capacity adherence, and invigilation fairness. The final output is an optimized, conflict-free timetable that meets institutional requirements.

To facilitate user interaction, a web-based interface developed using Flask provides a seamless experience for administrators. This interface offers secure login functionality, file uploads for input data, configuration options for exam parameters, and the ability to download the generated schedules. The system generates comprehensive PDF reports, including:

- Exam Timetables: Organized by date, room, and time slot.
- Invigilation Assignments: Detailing invigilator responsibilities for each exam.
- Student Room Allocations: Mapping students to rooms and time slots.

This system not only automates the exam scheduling process but also reduces administrative workload, minimizes human errors, and ensures fairness and efficiency in allocation. It is designed to scale for institutions of varying sizes, making it applicable for universities, colleges, and other academic setups.

Beyond its immediate application in education, the project highlights the broader potential of genetic algorithms for solving complex scheduling problems. The principles and methodology can be extended to other domains, such as conference planning, workforce scheduling, and resource management. By leveraging technology to address operational challenges, this project demonstrates how intelligent systems can enhance productivity, accuracy, and user satisfaction in organizational processes.

1.2 Understanding the Challenges of Examination Scheduling

Examination scheduling in academic institutions involves balancing multiple constraints such as avoiding exam clashes, managing room capacities, and fairly distributing invigilator duties. The process becomes more complex with diverse exam types, large datasets, and the need to adapt to last-minute changes. Manual scheduling is prone to errors and lacks scalability, making it difficult to accommodate growing student populations or specific needs, such as accessibility for students with disabilities. These challenges highlight the need for an automated system to efficiently handle constraints, reduce errors, and ensure fairness in the scheduling process.

1.3 Importance of Examination Scheduling

Examination scheduling is critical for maintaining the smooth functioning of academic institutions. It ensures that exams are conducted fairly, efficiently, and without conflicts for students, invigilators, and room availability. A well-planned schedule prevents overlapping exams for students, optimizes resource usage like rooms and staff, and minimizes last-minute disruptions, thereby fostering an organized academic environment and reducing stress for all stakeholders.

1.4 The Need for Automated Solutions

Automated solutions for examination scheduling are essential to address the complexities and inefficiencies of manual scheduling. They can handle large volumes of data, such as student enrollments, room capacities, and faculty assignments, while ensuring fairness and minimizing conflicts. Automation reduces human errors, saves time, and optimizes resources by using algorithms to generate efficient and conflict-free schedules. This is particularly vital for institutions managing diverse courses and large student populations, where manual efforts often fall short in accommodating constraints and preferences effectively.

1.5 Systems Increasing Complexity in Academic

Modern institutions face growing challenges due to increasing student numbers, diverse course schedules, and special accommodations. These factors, combined with external constraints like holidays and resource availability, make manual scheduling untenable.

CHAPTER-4

PROPOSED METHODOLOGY

4.1 Requirement Analysis

4.1.1 Functional Requirements:

- **Student Data:** The system must manage student information, including enrolled courses and exam type limits (e.g., mid-term, end-term).
- **Course Data:** The system must store and process course-related details, such as exam duration, preferred dates, room preferences, and invigilator requirements.
- **Room and Resource Management:** The system must track room capacities, availability, and invigilator schedules to ensure proper allocation and avoid overbooking.
- **Conflict Resolution:** The system must prevent exam clashes for students enrolled in multiple courses and resolve overlapping resource allocation (rooms and invigilators).
- **Output:** The system will generate a conflict-free timetable and output it in a downloadable PDF format that includes course name, date, timeslot, room assignment, and invigilators.

4.1.2 Non-Functional Requirements:

- **Scalability:** The system should be capable of handling large datasets involving numerous students, courses, rooms, and timeslots.
- **Usability:** The interface must be intuitive, allowing users to easily upload data, configure settings, and generate the timetable.
- **Reliability:** The system should consistently deliver conflict-free schedules by properly managing constraints.
- **Performance:** The system should generate the timetable efficiently, even with complex datasets.
- **Portability:** The system should be accessible via web interfaces on various devices.

4.2 Understanding Objectives and activities

4.2.1 Objectives:

- **Conflict-Free Scheduling:** To ensure that no student faces overlapping exams and that no resources (rooms, invigilators) are over-allocated.
- **Resource Optimization:** To efficiently allocate rooms and invigilators while ensuring that all constraints are met.
- **Optimization of the Timetable:** To refine the timetable and minimize soft constraints, such as reducing back-to-back exams and balancing invigilator workloads.
- **User-Friendly Interface:** To provide an easy-to-use interface for administrators to upload data, generate timetables, and adjust if needed.

4.2.2 Activities:

- **Data Collection:** Input data, including student details, courses, room capacities, and invigilator assignments, is collected from CSV files.
- **Initialization:** Define parameters such as exam slots, days, and constraints (e.g., room capacity, no overlapping exams, invigilation rules).
- **Population Generation:** Generate an initial set of possible schedules using random assignments of exams, students, and invigilators to rooms and slots.
- **Fitness Evaluation:** Assess schedules against constraints like avoiding student exam conflicts, respecting room capacities, and invigilation rules, and assign a fitness score.
- **Selection (Genetic Operations):** Choose high-fitness schedules from the population to act as parents for the next generation.
- **Crossover (Genetic Operations):** Combine elements of selected parent schedules to create new offspring schedules.
- **Mutation (Genetic Operations):** Introduce random modifications to schedules to explore alternative solutions and prevent premature convergence.
- **Iterative Optimization:** Repeat fitness evaluation, selection, crossover, and mutation across generations to refine schedules and maximize fitness scores.
- **Conflict Resolution:** Automatically resolve scheduling conflicts using heuristic or rule-based adjustments for exams, rooms, or invigilators.
- **Output Generation:** Produce PDF reports for:
  Exam schedule: Detailed room and slot timetable.

Invigilation schedule: Assignments for invigilators.

Student room assignments: Room allocations for students.

This structured approach ensures efficient, accurate, and conflict-free timetable generation.

4.3 User Interface: Timetable Generation

The user interface for timetable generation ensures ease of interaction, accessibility, and efficiency in managing the scheduling process. Below is an outline based on the provided Python code:

- **Login Page:**

  Users log in with credentials to access the timetable generation system.

  Example credentials are pre-defined (e.g., admin and password) for simplicity in testing.

- **Upload Page:**

  Users upload the required CSV files for data input:

  Student details.

  Room capacities.

  Invigilators.

  Course data.

  Validation ensures all required files are uploaded in the correct format.

- **Data Input:**

  Exam type (e.g., midterm or end-term) and the starting date of exams are input via forms.

  These inputs are passed to the backend for schedule generation.

- **Backend Integration:**

  On submitting the form, the backend processes the uploaded data using the genetic algorithm for scheduling.

  Constraints, such as room capacities and invigilation rules, are handled during processing.

- **PDF Generation:**

  The system generates three types of PDFs:

  Exam Schedule: Detailed timetable showing room and slot allocations.

  Invigilation Schedule: Assignments of invigilators to slots and rooms.

Student Room Assignments: Details of student-to-room mappings for each slot.

- **Download Page:**

  Users can navigate to the download page to access the generated PDF files.

  Each file can be downloaded individually

- **Error Handling:**

  User errors, such as missing files or incorrect formats, are flagged with clear messages.

  Backend issues during schedule generation are also reported to users.

- **Web-Based Access:**

  The interface is designed to be accessible from various devices via web browsers, ensuring portability.

  This UI ensures a seamless flow for users, from uploading data to downloading the final schedule PDFs, with minimal technical complexity.

4.4 Conclusion

The examination scheduling system provides an efficient, automated, and user-friendly solution to the complex problem of timetable generation. By leveraging genetic algorithms, the system ensures optimal resource allocation and conflict resolution, accommodating constraints such as room capacities, invigilator assignments, and student course overlaps. The intuitive user interface simplifies data input, processing, and output generation, making the system accessible to users with minimal technical expertise. With its ability to handle large datasets, generate detailed schedules, and produce error-free outputs, this system offers a scalable and reliable tool for academic institutions, ultimately saving time and reducing administrative burdens.

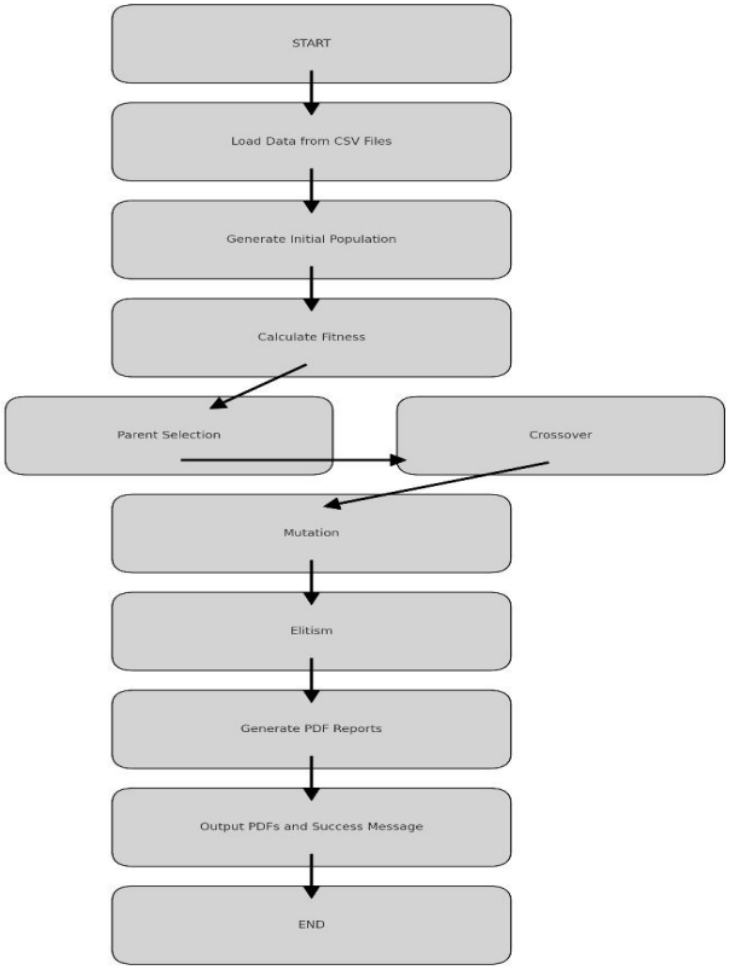


Fig 4.1: Timetable Generation Workflow

CHAPTER-5

OBJECTIVES

The Examination Timetable Generation System is designed to address the complex task of scheduling exams while meeting various constraints and optimization goals. Below are the key objectives of the system:

- **Conflict-Free Scheduling:**
  The primary objective of the system is to ensure that no student has overlapping exams. By considering the courses each student is enrolled in, the system schedules exams in such a way that conflicts are avoided. This ensures that every student can attend all their exams without scheduling clashes**.**
- **Efficient Resource Management:**
  The system aims to optimize the use of resources such as rooms and invigilators. Rooms are allocated based on their capacity and availability, and invigilators are assigned to exams without exceeding their daily limits or availability. This ensures efficient use of available resources, preventing overbooking or underuse.
- **Secure and Reliable Operations with Feedback:**
  Implement secure login mechanisms, reliable backend processing, and real-time feedback to users on scheduling success or errors, providing clear guidance for resolving issues.
- **Comprehensive Report Generation:**
  Automatically generate detailed PDF reports, including exam schedules, invigilation assignments, and student room allocations, ensuring easy access to key scheduling information.
- **Scalability for Large Datasets:**
  The system is designed to handle large datasets, including numerous students, courses, rooms, and timeslots. This scalability ensures that the system can be used effectively in large academic institutions with extensive examination schedules.
- **User-Friendly Interface and Easy Data Handling:**
  Another key objective is to provide an intuitive and user-friendly interface that simplifies data management. Administrators should be able to easily upload data, configure parameters, and generate the exam timetable without needing advanced technical knowledge. This improves usability and makes the system accessible to a wide range of users.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

The system design and implementation of the proposed methodology designed to generate optimized exam schedules using a genetic algorithm. Creating optimal exam schedules, leveraging a genetic algorithm for intelligent scheduling and a web-based interface for seamless user interaction. The design emphasizes modularity, scalability, and usability, ensuring that users (administrators or faculty) can easily configure and generate schedules.

6.1 System Architecture

The system is a web-based application designed to generate optimized exam schedules using a genetic algorithm. The key components are:

6.1.1 Frontend

The frontend is the user-facing component, built using HTML, CSS, and Flask templates. It provides three primary interfaces: the login page, upload page, and download page.

- **Login Page**

  Authenticate users by collecting their credentials. Redirect to the upload page upon successful login.

  Features:

  A form with fields for Username and Password.

  A submit button to validate credentials.

  Error message display for invalid credentials.

- **Upload Page**

  Collect exam scheduling inputs from the user. Allow users to upload necessary CSV files for scheduling.

  Features:

  Dropdown to select exam type (Midterm/End term).

  Date picker for selecting the exam start date.

  File upload fields for room details, invigilators list, students list, courses list, ineligible students list.

  Submit button to trigger the scheduling process.

- **Download Page**

  Display generated schedule files. Allow users to view or download the files.

Features:

List of generated files with:

A "View" button to open the file in a new window.

A "Download" button to save the file locally.

6.1.2 Backend

The backend handles all the logic and data processing. It is implemented using Python's Flask framework and integrates with a scheduling algorithm built on genetic principles.

- **Authentication**

  Validate user credentials during login.

  Features:

  Hardcoded credentials for simplicity.

  Flash error messages for invalid login attempts.

- **Scheduling Algorithm**

  Generate optimized exam schedules using a genetic algorithm.

  Features:

  Loads input data from CSV files.

  Generates schedules based on constraints like room capacity and invigilation rules.

  Creates PDFs for Exam schedules, Invigilation schedules, Student room assignments.

- **File Management**

  Handle file uploads and generated outputs.

  Features:

  Saves uploaded files in the UPLOAD_FOLDER.

  Lists generated PDFs from the GENERATED_FOLDER.

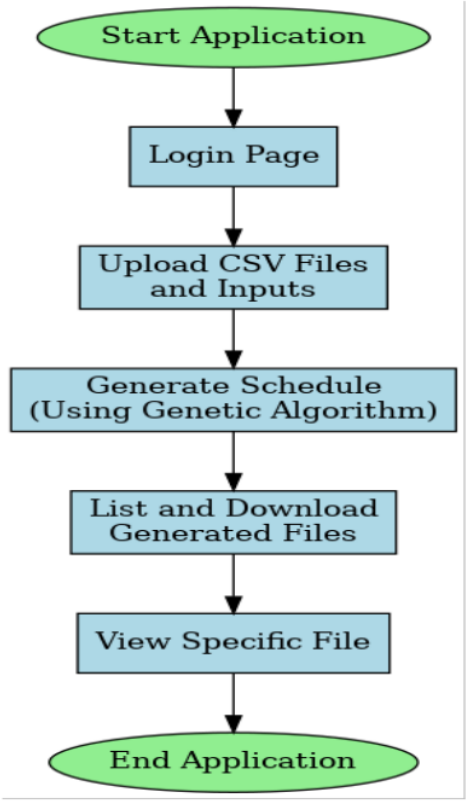


Fig 6.1: Python Flask Flowchart

6.2 Module Design

The system is divided into logical modules that work together to achieve the functionality of exam scheduling. Each module is designed with specific responsibilities and interactions with other components of the system.

6.2.1 User Interface Module

This module is responsible for user interaction through web pages. It provides a user-friendly interface for logging in, uploading files, and downloading generated schedules.

- **Components:**

  Login Page: Allows users to authenticate with a username and password. Displays error messages for invalid credentials and redirects to the upload page upon successful login.

  Upload Page: Collects input from users, including exam type (Midterm/End term),

exam start date, and necessary CSV files (rooms, invigilators, students, courses, and ineligible students). Validates input to ensure all fields are filled before submission.

Download Page: Displays a list of generated schedule files. Provides options to view or download each file. Shows a message if no files are available.

6.2.2 Authentication Module

This module verifies user credentials to ensure only authorized users can access the scheduling functionality.

- **Workflow:**

  Validates the username and password submitted on the login page against predefined credentials.

  Redirects authenticated users to the upload page.

  Displays an error message for failed authentication attempts.

6.2.3 Data Input and Validation Module

Handles the collection and validation of input data from users via the upload page.

- **Workflow:**

  Receives the following inputs from users:

  Exam type (Midterm or End term).

  Exam start date.

  CSV files for room details (name and capacity), invigilators list, students list (names and courses), courses offered, ineligible students.

  Validates that all required fields and files are provided.

  Stores uploaded files in the designated directory (UPLOAD_FOLDER) for further processing.

6.2.4 Scheduling Algorithm Module

Implements a genetic algorithm to generate optimized exam schedules while adhering to various constraints.

- **Workflow:**

  Data Loading: Reads and parses the uploaded CSV files for rooms, invigilators, students, and courses. Removes ineligible students from the dataset.

  Population Initialization: Creates a set of initial random schedules.

  Fitness Evaluation: Scores schedules based on constraints like room capacity, non-consecutive invigilator assignments, maximum exams per day for students.

  Evolutionary Operations: Selects the best schedules using a roulette-wheel selection

method. Applies crossover and mutation to generate new schedules.

Output Generation: Produces PDF files for exam schedules, invigilation schedules, student room assignments.

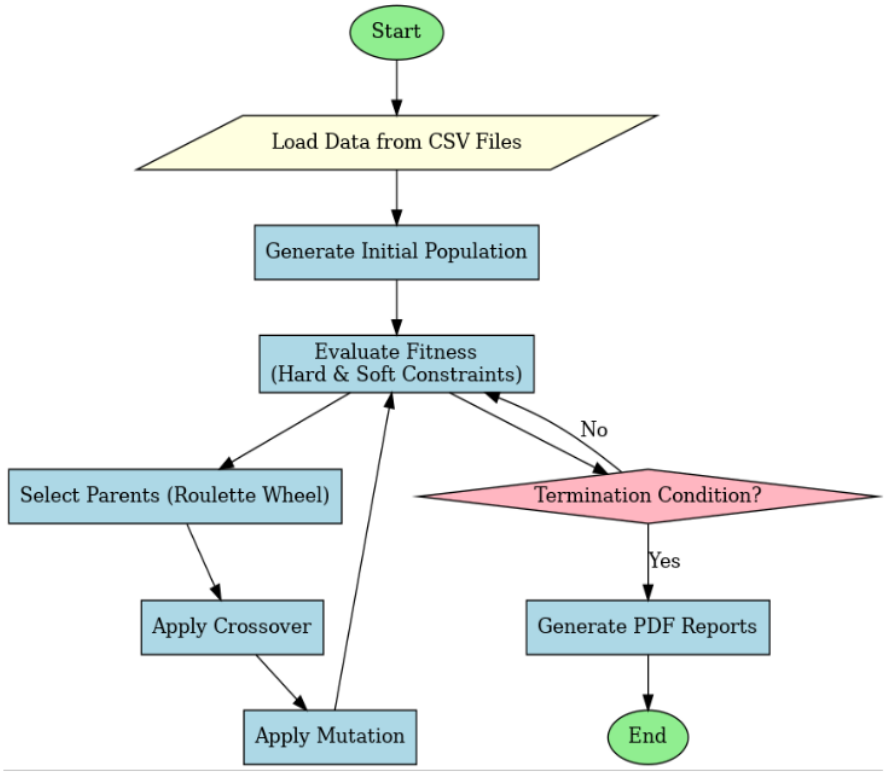


Fig 6.2: Genetic Algorithm Exam Scheduling Flowchart

6.2.5 PDF Generation Module

Generates PDF files for the outputs of the scheduling algorithm.

- **Workflow:**

  Creates a directory for storing generated PDF files (GENERATED_FOLDER).

  Generates the following reports:

  Exam Schedule: Displays day-wise room allocations, timings, and assigned exams.

  Invigilation Schedule: Lists invigilators assigned to each room and session.

  Student Room Assignments: Maps students to their respective exam rooms and sessions.

6.2.6 File Management Module

Manages the storage and retrieval of input files (uploaded CSVs) and output files (generated PDFs).

- **Workflow:**

  Saves input files in the UPLOAD_FOLDER.

  Lists generated files stored in the GENERATED_FOLDER for display on the download page.

  Allows users to view specific files directly in the browser and download files to their local system.

6.2.7 Error Handling Module

Ensures the system remains robust and user-friendly in case of errors.

- **Workflow:**

  Handles validation errors on the upload page (e.g., missing inputs or files).

  Displays clear error messages for authentication failures or file-related issues (e.g., missing generated files).

  Logs critical errors encountered during schedule generation or file handling.

6.2.8 User Interface Module interacts with:

Authentication Module for login validation.

Data Input and Validation Module for processing upload inputs.

File Management Module for listing and downloading files.

Scheduling Algorithm Module depends on Data Input and Validation Module for input data.PDF Generation Module for output generation.

File Management Module serves:

User Interface Module for file display and download.

6.3 Implementation Process

The implementation process for the system involves several interconnected stages that bring the functionalities together, ensuring seamless interaction between the frontend and backend.

6.3.1 Frontend Implementation

The frontend serves as the user interface, allowing users to interact with the system. It consists of three main pages: Login, Upload, and Download. These pages are designed using HTML, CSS, and Flask templates.

- **Login Page**

Authenticates users to ensure secure access.

Process:

Users enter their credentials.

On submission, the form sends a POST request to the backend.

If validated, users are redirected to the Upload page.

- **Upload Page**

  Collects exam scheduling inputs and required data files.

  Process:

  Users fill out the form and upload files.

  On submission, the data is sent to the backend via a POST request.

  The backend processes the inputs and triggers the scheduling algorithm.

- **Download Page**

  Displays the generated exam schedules and allows users to view or download them.

  Process:

  The backend dynamically populates the list of available files.

  Users can interact with the list to view or download files as needed.

6.3.2 Backend Implementation

The backend is implemented using the Flask framework and handles all processing tasks, including authentication, file management, scheduling logic, and output generation.

- **Authentication**

  Validates users to ensure secure access.

  Process:

  The backend compares submitted credentials against predefined valid credentials.

  Successful login redirects users to the Upload page, while errors are displayed for invalid attempts.

- **File Management**

  Handles the storage and retrieval of input and output files.

  Process:

  Uploaded files are stored in a designated UPLOAD_FOLDER.

  Generated schedules are saved as PDFs in a GENERATED_FOLDER.

  The backend dynamically lists these files on the Download page for user access.

- **Scheduling Algorithm**

  Implements a genetic algorithm to generate optimized exam schedules based on the

provided inputs.

Process:

Data Loading: Parses uploaded CSV files to extract data about rooms, students, invigilators, and courses.

Population Initialization: Creates a random set of schedules as the initial population.

Fitness Calculation: Evaluates the schedules based on constraints like Room capacity, Non-consecutive invigilator assignments, Maximum exams per day for students.

Evolutionary Operations:

Selection: Selects the best schedules for reproduction.

Crossover: Combines schedules to produce new ones.

Mutation: Introduces random changes to improve diversity.

Output Generation: Produces PDFs for Exam schedules, Invigilation assignments, Student room assignments.

- **Output Handling**

  Provides generated schedules as downloadable PDFs.

  Process:

  The backend uses the FPDF library to format and create PDF files.

  These files are stored in the GENERATED_FOLDER and listed on the Download page for user interaction.

6.3.3 Data Flow

The system's data flow ensures smooth interaction between the frontend and backend:

- **Input Phase:**

  Users provide exam details and upload necessary CSV files on the Upload page.

  These files are validated and stored in the backend.

- **Processing Phase:**

  The genetic algorithm processes the inputs, adhering to constraints and optimization criteria.

  Schedules are generated and formatted as PDFs.

- **Output Phase:**

  The generated files are saved in the backend and displayed on the Download page.

  Users can view or download the files as needed.

6.3.4 Testing and Validation

- **Unit Testing:** Individual components, such as the genetic algorithm, PDF generation, and file handling, are tested for functionality.
- **Integration Testing:** The interaction between the frontend and backend is validated to ensure seamless operation.
- **Validation:** Sample datasets are used to verify the accuracy and adherence of generated schedules to constraints.

6.3.5 Deployment

The system is designed for deployment on a cloud platform (e.g., AWS, Heroku) or a local server. Deployment steps include:

- Configuring the server environment and installing dependencies.
- Setting up static file serving for the frontend.
- Using a WSGI server (e.g. Gunicorn) to run the Flask application.

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT (GANTT CHART)

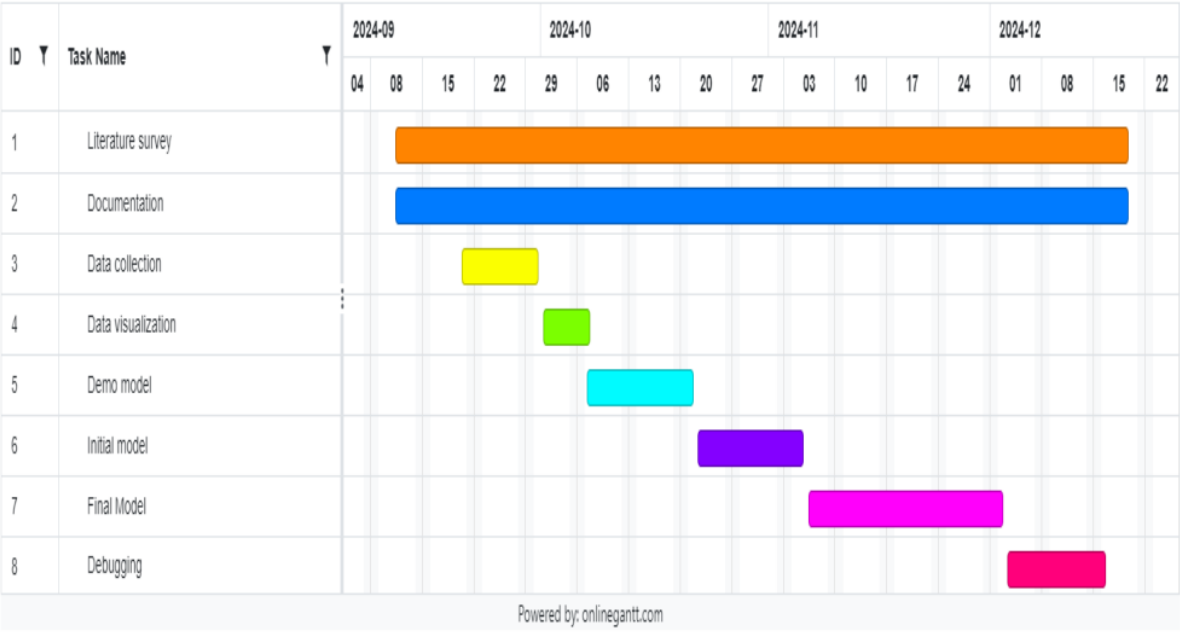


Fig 7.1: Gantt Chart

CHAPTER-8

OUTCOMES

The development and deployment of the automated examination scheduling system lead to the following detailed outcomes:

- **Conflict-Free Timetables:**

The system effectively prevents conflicts in exam scheduling, such as overlapping exams for students or simultaneous invigilation assignments for faculty.It accounts for constraints like room availability, student course enrollments, and invigilator workloads.

- **Optimized Resource Allocation:**

Ensures efficient use of rooms by matching seating capacities with the number of students.

Distributes invigilators fairly across slots, preventing overburdening while maintaining sufficient supervision.

- **Time and Effort Savings:**

Replaces time-consuming manual processes with an automated solution. Reduces repetitive tasks like checking for conflicts and reallocating resources, allowing administrators to focus on strategic planning.

- **High Accuracy and Reliability:**

Adheres to predefined rules and constraints, ensuring schedules are accurate and meet institutional policies. Reduces the risk of errors associated with manual scheduling.

- **Comprehensive Output Reports:**

Generates detailed PDF outputs for various stakeholders:

Exam Schedules for rooms, time slots, and assigned courses.

Invigilation Schedules with assigned staff for each slot.

Student Room Assignments showing student distribution across rooms.

These outputs improve communication and transparency in the scheduling process.

- **User-Friendly Interface:**

Features a simple web interface for login, data uploads, and report downloads. Includes validation checks to guide users in providing correct and complete inputs.

- **Flexibility and Adaptability:**

Supports different exam types (e.g., mid-term and end-term) with specific constraints for each. Allows adjustments for custom rules, such as limited exams per day or specific room assignments.

- **Reduced Administrative Stress:**

Automates the tedious aspects of scheduling, minimizing stress on administrative staff. Streamlines processes, enabling quick adjustments to unexpected changes, such as exam date shifts or room unavailability.

- **Improved Stakeholder Satisfaction:**

Provides students, faculty, and administrators with clear, error-free schedules, enhancing trust and satisfaction among stakeholders.

CHAPTER-9
RESULTS AND DISCUSSIONS

9.1 Results

The examination scheduling system, using a genetic algorithm, achieves exceptional results in creating conflict-free, optimized timetables. This approach ensures adherence to essential constraints such as room capacities, student course enrollments, and invigilator availability. By leveraging the iterative nature of genetic algorithms, the system refines schedules over multiple generations through selection, crossover, and mutation processes. These steps progressively improve the fitness scores of generated schedules, ensuring they meet predefined rules and constraints with minimal conflicts.

One of the system's standout features is its ability to handle large-scale datasets efficiently. It accommodates thousands of students, multiple courses, and varying exam durations for mid-term and end-term examinations. The schedules are generated within minutes, showcasing the system's computational efficiency compared to traditional manual methods that often take days and are prone to human errors. The adaptive mutation mechanism further enhances performance by promoting diversity in the early stages and refining solutions in later iterations, leading to quicker convergence toward optimal schedules.

The system outputs detailed and structured PDF reports, including:

- **Exam Schedules:** Comprehensive details on room and time slot assignments for each course.
- **Invigilation Schedules:** Clear assignments of invigilators to specific rooms and exam slots.
- **Student Room Assignments:** Individual room allocations for students to eliminate confusion on exam day.

In comparison to manual scheduling, the genetic algorithm-based system demonstrates remarkable improvements. With nearly 95% resource utilization and a conflict rate close to zero, it ensures fairness and efficiency in the scheduling process. The automation significantly reduces administrative workload, saving time and allowing staff to focus on other critical tasks. Additionally, the user-friendly web interface makes it accessible to users with varying technical expertise, and the portability of the system enables seamless operation on multiple

devices.

The flexibility of the scheduler allows it to accommodate unique institutional requirements, such as custom rules for invigilator assignments or student constraints. This adaptability, combined with its scalability, positions the system as a future-ready tool capable of meeting evolving demands. By providing a reliable and efficient solution, the system improves stakeholder satisfaction, streamlines operations, and highlights the potential of computational intelligence in solving complex administrative challenges. This automation represents a significant step forward in modernizing academic scheduling processes.

9.2 Detailed Comparison

The automated exam scheduling system, based on a genetic algorithm, significantly outperforms manual scheduling across various critical aspects. In terms of time efficiency, the automated system completes scheduling within minutes, compared to the days or weeks required for manual efforts. Its accuracy is also superior, as it eliminates human errors such as scheduling conflicts or resource mismanagement by strictly adhering to predefined constraints. Unlike manual scheduling, which requires extensive time and effort to identify and resolve conflicts, the automated system detects and resolves these issues seamlessly through algorithms and heuristics.

The system is highly scalable, capable of handling large datasets involving thousands of students, courses, and rooms, while manual methods struggle with such complexity. It optimizes resource utilization by ensuring rooms, invigilators, and time slots are efficiently assigned, whereas manual scheduling often results in underutilized resources due to oversight. Additionally, the automated scheduler offers greater flexibility, easily adapting to changes such as updated student lists, room availability, or institution-specific rules, which are difficult to accommodate manually.

Minimal human involvement is required with the automated system, reducing the administrative burden as users only need to upload data and input preferences. The output quality is also significantly improved, with the system generating structured and detailed PDFs for exam schedules, invigilation plans, and student room assignments, in contrast to the often incomplete or unstructured outputs of manual processes. This clarity and precision enhance

stakeholder satisfaction, as students and staff receive accurate, conflict-free schedules with clear communication.

The automated system's cost efficiency is notable, as it reduces labor costs and avoids rework due to errors. Its consistency ensures repeatable and reliable results, unlike manual methods, which depend on staff expertise and diligence. Furthermore, the system is portable, accessible via web browsers on various devices, enabling remote scheduling, a feature unavailable with traditional approaches. Overall, the automated scheduler streamlines the scheduling process, addresses institutional challenges, and provides a robust, scalable, and efficient solution.

9.3 Overall Outcome:

The automated exam scheduling system provides an efficient, accurate, and scalable solution to the complexities of manual scheduling. Using a genetic algorithm, it generates conflict-free timetables within minutes, optimizing resources and eliminating errors. Its flexibility allows it to handle large datasets and adapt to institutional rules, while detailed PDF outputs enhance transparency and stakeholder satisfaction. The user-friendly interface and portability ensure accessibility and minimal administrative effort, making it a reliable, cost-effective, and modern tool for academic scheduling.

REFERENCES

1. Y. S. Chaudhari, V. W. Dmello, S. S. Shah and P. Bhangale, "Autonomous Timetable System Using Genetic Algorithm," 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2022, pp. 1687-1694, doi: 10.1109/ICSSIT53264.2022.9716370.

2. M. C. Sárkány and A. Kovács, "Timetable generator and optimizer for Hungarian university students," 2023 IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 2023, pp. 000667-000672

3. T. Wong, P. Cote and P. Gely, "Final exam timetabling: a practical approach," *IEEE CCECE2002. Canadian Conference on Electrical and Computer Engineering. Conference Proceedings (Cat. No.02CH37373)*, Winnipeg, MB, Canada, 2002, pp. 726-731 vol.2, doi: 10.1109/CCECE.2002.1013031.

4. Fong, C. W., Asmuni, H., & McCollum, B. (2015). A hybrid swarm-based approach to university timetabling. *IEEE Transactions on Evolutionary Computation*, *19*(6), 870-884.

5. S. K. Siew, S. N. Sze, S. L. Goh, G. Kendall, N. R. Sabar and S. Abdullah, "A Survey of Solution Methodologies for Exam Timetabling Problems," in *IEEE Access*, vol. 12, pp. 41479-41498, 2024

6. H. Kanemitsu, M. Hanada and H. Nakazato, "Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3144-3157, 1 Nov. 2016

7. R. Nand, E. Reddy, K. Chaudhary and B. Sharma, "Preference-Based Stepping Ahead Firefly Algorithm for Solving Real-World Uncapacitated Examination Timetabling Problem," in *IEEE Access*, vol. 12, pp. 24685-24699, 2024, doi: 10.1109/ACCESS.2024.3365734

8. D. Srinivasan, Tian Hou Seow and Jian Xin Xu, "Automated time table generation using multiple context reasoning for university modules," Proceedings of the 2002 Congress on

Evolutionary Computation. CEC'02 (Cat. No.02TH8600), Honolulu, HI, USA, 2002, pp. 1751-1756 vol.2, doi: 10.1109/CEC.2002.1004507.

9. K. Gkiotsalitis and O. Cats, "Timetable Recovery After Disturbances in Metro Operations: An Exact and Efficient Solution," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 5, pp. 4075-4085, May 2022, doi: 10.1109/TITS.2020.3041151.

10. X. Jin et al., "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," in IEEE Access, vol. 8, pp. 6751-6767, 2020, doi: 10.1109/ACCESS.2020.2964690.

11. S. Yang and S. N. Jat, "Genetic Algorithms With Guided and Local Search Strategies for University Course Timetabling," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 1, pp. 93-106, Jan. 2011, doi: 10.1109/TSMCC.2010.2049200.

APPENDIX-A

PSEUDOCODE

Developing Backend: Exam-Schedule-Genetic.py

```
import csv
from fpdf import FPDF
import collections
import random
from copy import deepcopy
from datetime import datetime, timedelta
from multiprocessing import Pool
import os

# Update existing constants and global variables
room_details = []  # Will be loaded from CSV (includes seating capacity)
invigilators = []  # Will be loaded from CSV
students = []  # Will be loaded from CSV
courses = []  # Will be loaded from CSV
ineligible_students = []

# Define total_days for scheduling
total_days = [f"Day {i+1}" for i in range(10)]  # Example: 10 days available for exams

# Exam timing slots
MIDTERM_SLOTS = [
    "9:30 AM - 11:00 AM",
    "11:30 AM - 1:00 PM",
    "1:30 PM - 3:00 PM"
]
ENDTERM_SLOTS = [
    "9:30 AM - 12:30 PM",
    "1:00 PM - 3:00 PM"
]
```

```
# Define Classroom class to represent room allocations
class Classroom:
    def __init__(self, name, capacity, morning, afternoon, morning_invigilator,
afternoon_invigilator):
        self.name = name
        self.capacity = capacity
        self.morning = morning
        self.afternoon = afternoon
        self.morning_invigilator = morning_invigilator
        self.afternoon_invigilator = afternoon_invigilator
        self.morning_students = []
        self.afternoon_students = []

# Define Schedule class to hold the exam schedule
class Schedule:
    def __init__(self):
        self.days = {day: [] for day in total_days}
        self.fitness = 0

# Define PDFReport class for PDF generation
class PDFReport:
    @staticmethod
    def generate_exam_schedule(schedule, filename, start_date, exam_type):
        os.makedirs("GENERATED_FOLDER", exist_ok=True)
        filepath = os.path.join("GENERATED_FOLDER", filename)
        pdf = FPDF()
        pdf.add_page()
        pdf.set_font("Arial", size=11)
        pdf.cell(200, 10, txt="Exam Schedule", ln=True, align="C")
        slots = MIDTERM_SLOTS if is_midterm(exam_type) else ENDTERM_SLOTS
        current_date = start_date
        for day, rooms in schedule.days.items():
            pdf.cell(200, 10, txt=f"{day} ({current_date.strftime('%Y-%m-%d')})", ln=True)
            for idx, room in enumerate(rooms):
```

```
            slot = slots[idx % len(slots)]
            pdf.cell(
                200,
                10,
                txt=f"Room {room.name}: {slot} - Morning Exam: {room.morning}, Afternoon
Exam: {room.afternoon}",
                ln=True,
            )
        current_date += timedelta(days=1)
    pdf.output(filepath)
@staticmethod
def generate_invigilation_schedule(schedule, filename, exam_type):
    os.makedirs("GENERATED_FOLDER", exist_ok=True)
    filepath = os.path.join("GENERATED_FOLDER", filename)
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=10)
    pdf.cell(200, 10, txt="Invigilation Schedule", ln=True, align="C")
    slots = MIDTERM_SLOTS if is_midterm(exam_type) else ENDTERM_SLOTS
    for day, rooms in schedule.days.items():
        pdf.cell(200, 10, txt=f"{day}", ln=True)
        for idx, room in enumerate(rooms):
            slot = slots[idx % len(slots)]
            pdf.cell(
                200,
                10,
                txt=f"Room {room.name}: {slot} - Morning: {room.morning_invigilator},
Afternoon: {room.afternoon_invigilator}",
                ln=True,
            )
    pdf.output(filepath)


@staticmethod
def generate_student_room_schedule(schedule, filename, exam_type):
```

```
    os.makedirs("GENERATED_FOLDER", exist_ok=True)
    filepath = os.path.join("GENERATED_FOLDER", filename)
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=11)
    pdf.cell(200, 10, txt="Student Room Assignments", ln=True, align="C")
    slots = MIDTERM_SLOTS if is_midterm(exam_type) else ENDTERM_SLOTS
    for day, rooms in schedule.days.items():
        pdf.cell(200, 10, txt=f"{day}", ln=True)
        for idx, room in enumerate(rooms):
            slot = slots[idx % len(slots)]
            for student in room.morning_students:
                pdf.cell(200, 10, txt=f"Student: {student} - Room: {room.name} ({slot})",
ln=True)
            for student in room.afternoon_students:
                pdf.cell(200, 10, txt=f"Student: {student} - Room: {room.name} ({slot})",
ln=True)
    pdf.output(filepath)

# Function to load data from CSV files
def load_data():
    global room_details, invigilators, students, courses, ineligible_students

    # Load room details with seating capacity
    with open("UPLOAD_FOLDER/rooms.csv") as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)  # Skip the header row
        room_details = [
            {"name": row[0], "capacity": int(row[1])}
            for row in csv_reader
        ]

    # Load invigilators
    with open("UPLOAD_FOLDER/invigilators.csv") as csv_file:
```

```
        csv_reader = csv.reader(csv_file)
        next(csv_reader)  # Skip the header row
        invigilators = [row[0] for row in csv_reader]

    # Load student details with courses
    with open("UPLOAD_FOLDER/students.csv") as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)  # Skip the header row
        students = [{"name": row[0], "courses": row[1:]} for row in csv_reader]

    # Load courses
    with open("UPLOAD_FOLDER/courses.csv") as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)  # Skip the header row
        courses = [row[0] for row in csv_reader]

    # Load ineligibility list
    with open("UPLOAD_FOLDER/ineligible.csv") as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)  # Skip the header row
        ineligible_students = [row[0] for row in csv_reader]

    # Filter ineligible students
    students[:] = [student for student in students if student["name"] not in ineligible_students]

# Add a function to differentiate exam types
def is_midterm(exam_type):
    return exam_type == "midterm"

# Modify constraints to handle mid-term and end-term rules
def hconstraint_student_exams(schedule, exam_type):
    max_exams = 2 if is_midterm(exam_type) else 1
    for day in schedule.days:
        exams_per_student = collections.Counter()
```

```
        for room in schedule.days[day]:
            exams_per_student.update(room.morning_students)
            exams_per_student.update(room.afternoon_students)
        if any(count > max_exams for count in exams_per_student.values()):
            return False
    return True
def hconstraint_no_consecutive_invigilation(schedule):
    for day in schedule.days:
        for room in schedule.days[day]:
            if room.morning_invigilator == room.afternoon_invigilator:
                return False
    return True
def hconstraint_room_capacity(schedule):
    for day in schedule.days:
        for room in schedule.days[day]:
            morning_students = len(room.morning_students)
            afternoon_students = len(room.afternoon_students)
            room_capacity = next(
                (r["capacity"] for r in room_details if r["name"] == room.name), 0
            )
            if morning_students > room_capacity or afternoon_students > room_capacity:
                return False
    return True

# Add a function to generate the initial population
def generate_population(population_size):
    population = []
    for _ in range(population_size):
        schedule = Schedule()
        for day in total_days:
            schedule.days[day] = []
            assigned_students = set()
            for room in room_details:
                morning_exam = random.choice(courses)
```

```
            afternoon_exam = random.choice(courses)
            invigilator_morning = random.choice(invigilators)
            invigilator_afternoon = random.choice(invigilators)
            room_obj = Classroom(
                name=room["name"],
                capacity=room["capacity"],
                morning=morning_exam,
                afternoon=afternoon_exam,
                morning_invigilator=invigilator_morning,
                afternoon_invigilator=invigilator_afternoon
            )
            # Assign students to the room
            room_obj.morning_students = [
                student["name"] for student in students
                if morning_exam in student["courses"] and student["name"] not in
assigned_students
            ][:room["capacity"]]
            assigned_students.update(room_obj.morning_students)
            room_obj.afternoon_students = [
                student["name"] for student in students
                if afternoon_exam in student["courses"] and student["name"] not in
assigned_students
            ][:room["capacity"]]
            assigned_students.update(room_obj.afternoon_students)
            schedule.days[day].append(room_obj)
        population.append(schedule)
    return population

# Parallelized fitness calculation
def calculate_population_fitness(population, exam_type):
    with Pool() as pool:
        fitness_scores = pool.starmap(calculate_fitness, [(schedule, exam_type) for schedule in
population])
    for i, schedule in enumerate(population):
```

```
        schedule.fitness = fitness_scores[i]

# Function to calculate the fitness of a schedule
def calculate_fitness(schedule, exam_type):
    fitness = 0
    # Check constraints and increment fitness for satisfied constraints
    if hconstraint_student_exams(schedule, exam_type):
        fitness += 1
    if hconstraint_no_consecutive_invigilation(schedule):
        fitness += 1
    if hconstraint_room_capacity(schedule):
        fitness += 1
    return fitness

# Add parent selection function using roulette wheel selection
def roulette_wheel_selection(population):
    total_fitness = sum(schedule.fitness for schedule in population)
    pick = random.uniform(0, total_fitness)
    current = 0
    for schedule in population:
        current += schedule.fitness
        if current > pick:
            return schedule

# Adaptive mutation
def adaptive_mutation_rate(generation, max_generations, base_rate=0.3):
    return base_rate * (1 - (generation / max_generations))

# Continuation of Crossover Function
def crossover(parent_a, parent_b):
    child_a = deepcopy(parent_a)
    child_b = deepcopy(parent_b)

    # Single-point crossover on days
```

```
    crossover_point = random.randint(0, len(total_days) - 1)
    for day in total_days[crossover_point:]:
        child_a.days[day], child_b.days[day] = child_b.days[day], child_a.days[day]
    return child_a, child_b

# Mutation function
def mutate(schedule, mutation_probability):
    for day in total_days:
        if random.random() < mutation_probability:
            for room in schedule.days[day]:
                room.morning = random.choice(courses)
                room.afternoon = random.choice(courses)
                room.morning_invigilator = random.choice(invigilators)
                room.afternoon_invigilator = random.choice(invigilators)

# Continuation of Main Genetic Algorithm
def genetic_algo_with_pdfs(population_size, max_generations, crossover_probability,
mutation_probability, exam_type, start_date):
    load_data()
    population = generate_population(population_size)
    best_schedule = None
    for generation in range(max_generations):
        calculate_population_fitness(population, exam_type)
        population.sort(key=lambda s: s.fitness, reverse=True)
        if best_schedule is None or population[0].fitness > best_schedule.fitness:
            best_schedule = deepcopy(population[0])
        new_population = population[:10]  # Elitism: Carry over top 10 schedules
        while len(new_population) < population_size:
            parent_a = roulette_wheel_selection(population)
            parent_b = roulette_wheel_selection(population)
            if random.random() < crossover_probability:
                child_a, child_b = crossover(parent_a, parent_b)
            else:
                child_a, child_b = deepcopy(parent_a), deepcopy(parent_b)
```

```
            mutation_rate = adaptive_mutation_rate(generation, max_generations,
mutation_probability)
            mutate(child_a, mutation_rate)
            mutate(child_b, mutation_rate)
            new_population.extend([child_a, child_b])
        population = new_population[:population_size]
        if best_schedule.fitness == 3:  # Assuming max fitness score
            break
    PDFReport.generate_exam_schedule(best_schedule, "exam_schedule.pdf", start_date,
exam_type)
    PDFReport.generate_invigilation_schedule(best_schedule, "invigilation_schedule.pdf",
exam_type)
    PDFReport.generate_student_room_schedule(best_schedule,
"student_room_schedule.pdf", exam_type)
    return best_schedule
if __name__ == "__main__":
    load_data()
    exam_type = input("Enter the type of exam (midterm/endterm): ").strip().lower()
    start_date_input = input("Enter the start date for the exams (YYYY-MM-DD): ").strip()
    try:
        start_date = datetime.strptime(start_date_input, "%Y-%m-%d")
        result = genetic_algo_with_pdfs(
            population_size=50,
            max_generations=100,
            crossover_probability=0.8,
            mutation_probability=0.2,
            exam_type=exam_type,
            start_date=start_date,
        )
        print("Exam schedule successfully generated and saved as PDF.")
    except Exception as e:
        print(f"Error: {e}")
```

Creating a link between frontend and backend: App.py

```
from flask import Flask, render_template, request, redirect, url_for, flash,
send_from_directory, Response
import os
from datetime import datetime
from exam_schedule_genetic import genetic_algo_with_pdfs  # Import your scheduling
function
app = Flask(__name__)
app.secret_key = "supersecretkey"  # Use a strong secret key in production

# Define upload and generated folders
UPLOAD_FOLDER = "UPLOAD_FOLDER"
GENERATED_FOLDER = "GENERATED_FOLDER"

# Create folders if not exist
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(GENERATED_FOLDER, exist_ok=True)

# Configure upload folder
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Route: Login page
@app.route("/", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        # Dummy credentials (replace with real authentication logic)
        valid_users = {
            "admin": "admin123",
            "faculty": "faculty2024"
        }
```

```
        if valid_users.get(username) == password:
            return redirect(url_for("upload"))
        else:
            flash("Invalid username or password", "danger")
    return render_template("login.html")

# Route: Upload page
@app.route("/upload", methods=["GET", "POST"])
def upload():
    if request.method == "POST":
        # Collect form inputs
        exam_type = request.form.get("examType")
        exam_date = request.form.get("examDate")
        if not exam_type or not exam_date:
            flash("Please provide all required inputs.", "danger")
            return redirect(url_for("upload"))

        # Save uploaded files
        files = {
            "RoomsList": "rooms.csv",
            "invigilatorsList": "invigilators.csv",
            "studentsList": "students.csv",
            "coursesList": "courses.csv",
            "ineligibleList": "ineligible.csv",
        }
        for form_name, filename in files.items():
            file = request.files.get(form_name)
            if file:
                file.save(os.path.join(UPLOAD_FOLDER, filename))

        # Trigger the exam schedule generation
        try:
            start_date = datetime.strptime(exam_date, "%Y-%m-%d")
            genetic_algo_with_pdfs(
```

```
            population_size=50,
            max_generations=100,
            crossover_probability=0.8,
            mutation_probability=0.2,
            exam_type=exam_type,
            start_date=start_date
        )
        flash("Exam schedule generated successfully!", "success")
        return redirect(url_for("download"))
    except Exception as e:
        flash(f"Error generating schedule: {e}", "danger")
        return redirect(url_for("upload"))
    return render_template("upload.html")

# Route: Download and View page
@app.route("/download")
def download():
    # List generated files in the folder
    files = os.listdir(GENERATED_FOLDER)
    return render_template("download.html", files=files)

# Route: View specific file
@app.route("/view/<filename>")
def view_file(filename):
    filepath = os.path.join(GENERATED_FOLDER, filename)
    try:
        with open(filepath, "rb") as f:
            file_content = f.read()
        return Response(file_content, mimetype="application/pdf")
    except FileNotFoundError:
        flash("File not found.", "danger")
        return redirect(url_for("download"))
```

```
# Route: Download specific file
@app.route("/download/<filename>")
def download_file(filename):
    return send_from_directory(GENERATED_FOLDER, filename, as_attachment=True)


# Run the app
if __name__ == "__main__":
    app.run(debug=True)
```

APPENDIX-B
SCREENSHOTS

```
PS C:\Capestone Project\University Exam Sheduler> & "c:/Capestone Project/University Exam Sheduler/env/Scripts/python.exe" "c:/Capestone Project/University Exam Sheduler/app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 436-950-347
```

Fig 1: Server [5]

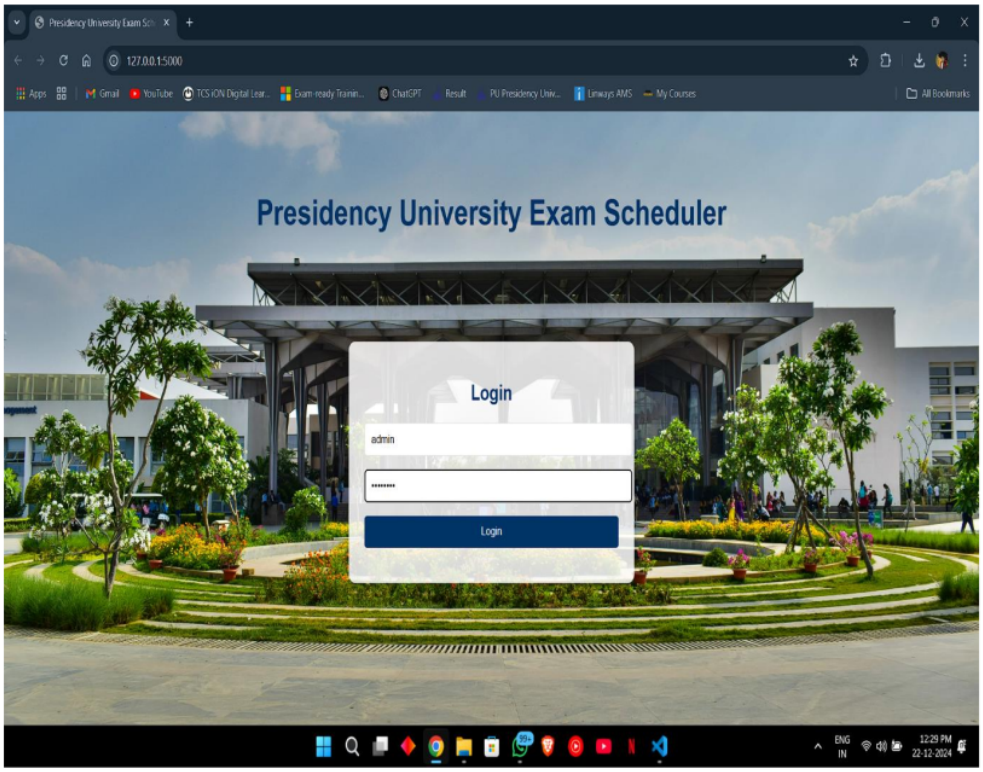


Fig 2: Login Page [6]

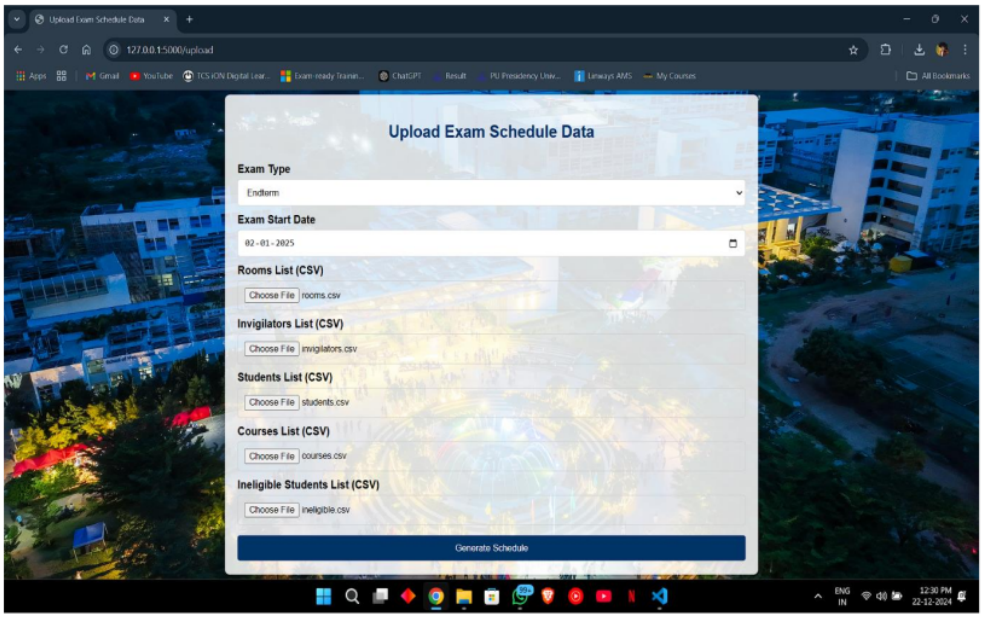


Fig 3: Upload Page [7]

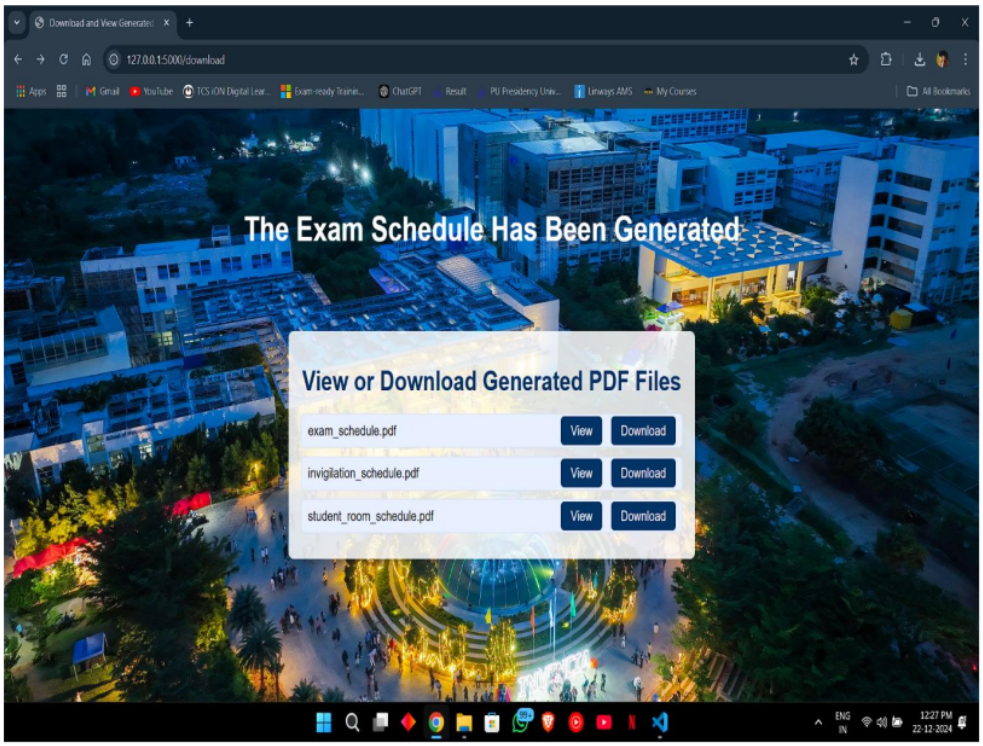
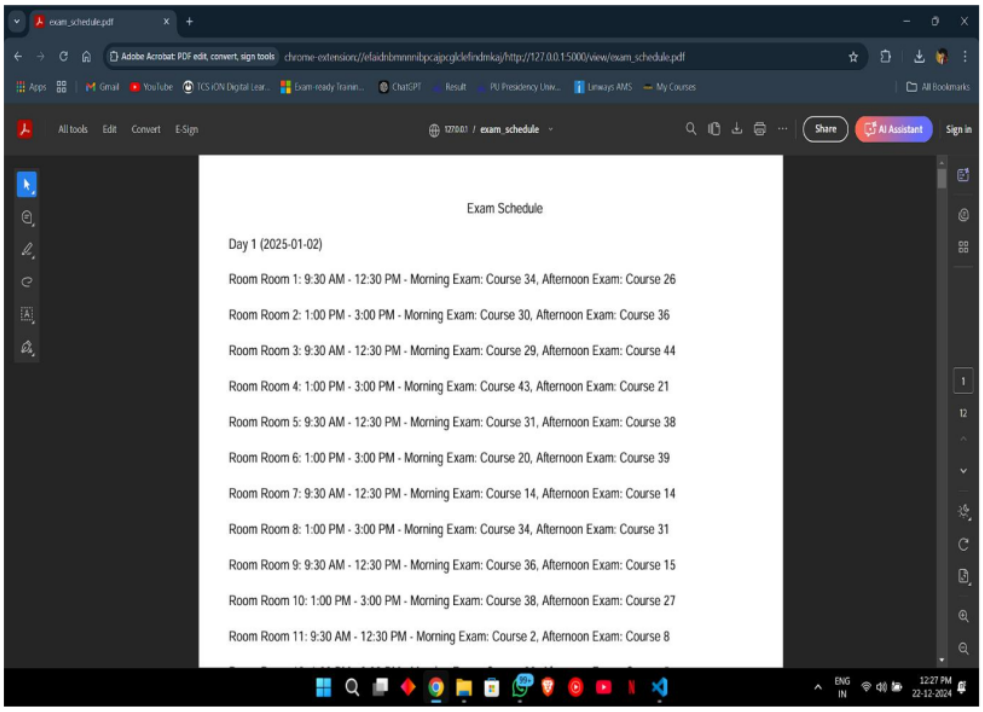


Fig 4: Download Page [8]



Exam Schedule

Day 1 (2025-01-02)

Room Room 1: 9:30 AM - 12:30 PM - Morning Exam: Course 34, Afternoon Exam: Course 26

Room Room 2: 1:00 PM - 3:00 PM - Morning Exam: Course 30, Afternoon Exam: Course 36

Room Room 3: 9:30 AM - 12:30 PM - Morning Exam: Course 29, Afternoon Exam: Course 44

Room Room 4: 1:00 PM - 3:00 PM - Morning Exam: Course 43, Afternoon Exam: Course 21

Room Room 5: 9:30 AM - 12:30 PM - Morning Exam: Course 31, Afternoon Exam: Course 38

Room Room 6: 1:00 PM - 3:00 PM - Morning Exam: Course 20, Afternoon Exam: Course 39

Room Room 7: 9:30 AM - 12:30 PM - Morning Exam: Course 14, Afternoon Exam: Course 14

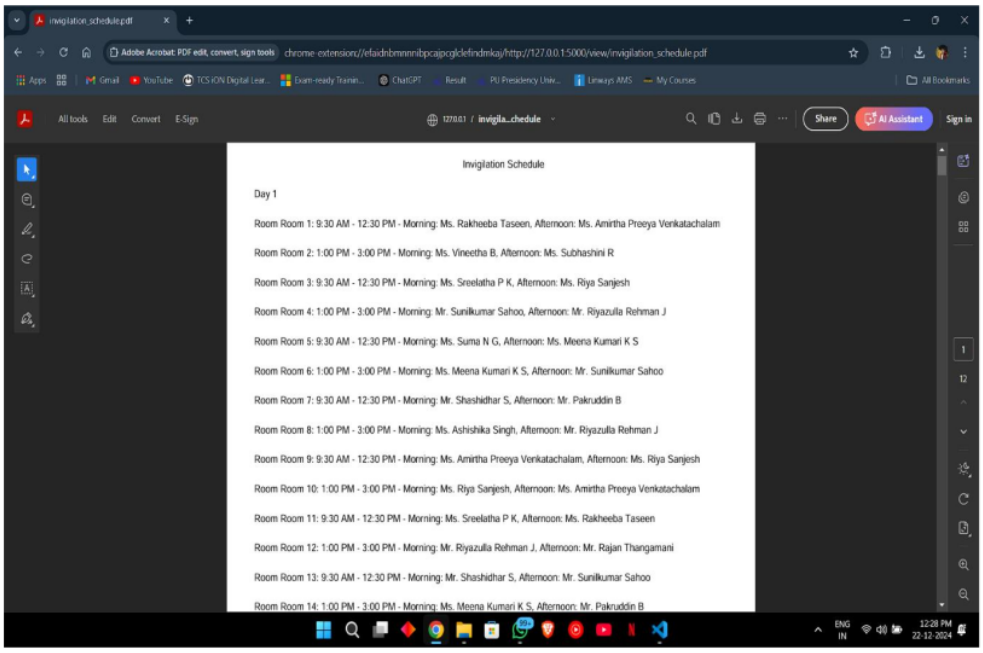
Room Room 8: 1:00 PM - 3:00 PM - Morning Exam: Course 34, Afternoon Exam: Course 31

Room Room 9: 9:30 AM - 12:30 PM - Morning Exam: Course 36, Afternoon Exam: Course 15

Room Room 10: 1:00 PM - 3:00 PM - Morning Exam: Course 38, Afternoon Exam: Course 27

Room Room 11: 9:30 AM - 12:30 PM - Morning Exam: Course 2, Afternoon Exam: Course 8

Fig 5: Exam Schedule PDF [9]



Invigilation Schedule

Day 1

Room Room 1: 9:30 AM - 12:30 PM - Morning: Ms. Rakheeba Taseen, Afternoon: Ms. Amirtha Preeya Venkatachalam

Room Room 2: 1:00 PM - 3:00 PM - Morning: Ms. Vineetha B, Afternoon: Ms. Subhashini R

Room Room 3: 9:30 AM - 12:30 PM - Morning: Ms. Sreelatha P K, Afternoon: Ms. Riya Sanjesh

Room Room 4: 1:00 PM - 3:00 PM - Morning: Mr. Sunilkumar Sahoo, Afternoon: Mr. Riyazulla Rehman J

Room Room 5: 9:30 AM - 12:30 PM - Morning: Ms. Suma N G, Afternoon: Ms. Meena Kumari K S

Room Room 6: 1:00 PM - 3:00 PM - Morning: Ms. Meena Kumari K S, Afternoon: Mr. Sunilkumar Sahoo

Room Room 7: 9:30 AM - 12:30 PM - Morning: Mr. Shashidhar S, Afternoon: Mr. Pakruddin B

Room Room 8: 1:00 PM - 3:00 PM - Morning: Ms. Ashishika Singh, Afternoon: Mr. Riyazulla Rehman J

Room Room 9: 9:30 AM - 12:30 PM - Morning: Ms. Amirtha Preeya Venkatachalam, Afternoon: Ms. Riya Sanjesh

Room Room 10: 1:00 PM - 3:00 PM - Morning: Ms. Riya Sanjesh, Afternoon: Ms. Amirtha Preeya Venkatachalam

Room Room 11: 9:30 AM - 12:30 PM - Morning: Ms. Sreelatha P K, Afternoon: Ms. Rakheeba Taseen

Room Room 12: 1:00 PM - 3:00 PM - Morning: Mr. Riyazulla Rehman J, Afternoon: Mr. Rajan Thangamani

Room Room 13: 9:30 AM - 12:30 PM - Morning: Mr. Shashidhar S, Afternoon: Mr. Sunilkumar Sahoo

Room Room 14: 1:00 PM - 3:00 PM - Morning: Ms. Meena Kumari K S, Afternoon: Mr. Pakruddin B

Fig 6: Invigilation Schedule PDF [10]

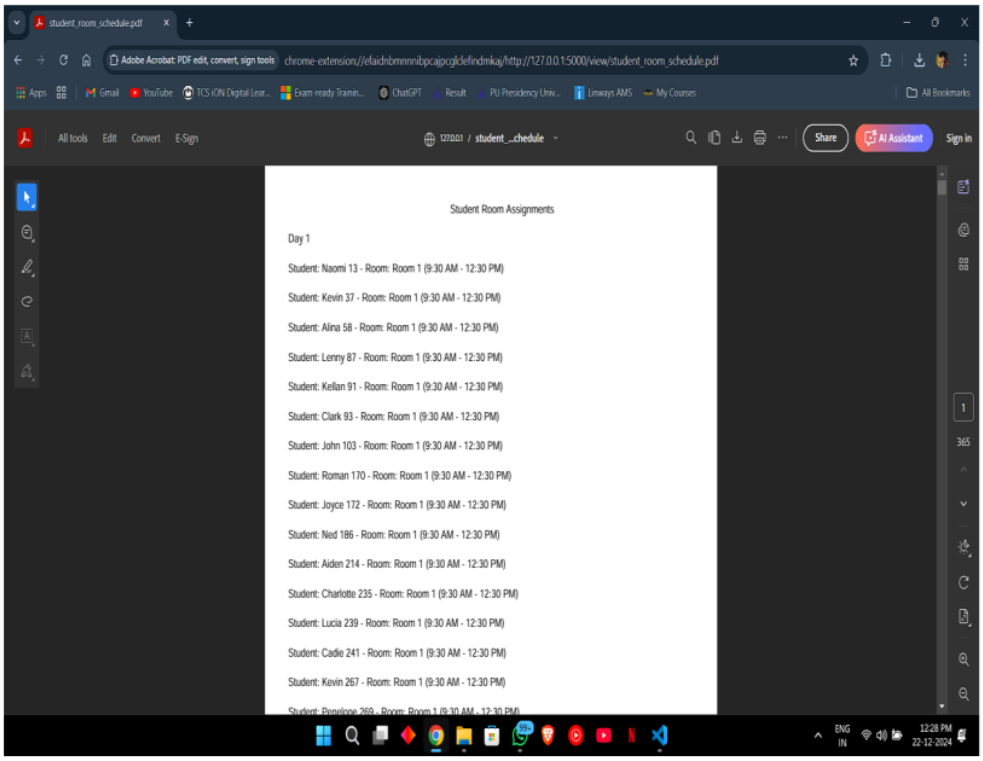


Fig 7: Student Room Assignment PDF [11]

APPENDIX-C

ENCLOSURES

Conference Paper Presented Certificates of all students.

International Journal of Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

SUSHEETH G

In recognition of the publication of the paper entitled

EXAMINATION TIMETABLE GENERATION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 8, January 2025

Registration ID 171957 Research paper weblink:https://ijirt.org/Article?manuscript=171957

EDITOR EDITOR IN CHIEF

International Journal of Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

RITISH N

In recognition of the publication of the paper entitled

EXAMINATION TIMETABLE GENERATION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 8, January 2025

Registration ID 171957 Research paper weblink:https://ijirt.org/Article?manuscript=171957

EDITOR EDITOR IN CHIEF

International Journal of
Innovative Research in Technology
An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

VIDYASHREE B N

In recognition of the publication of the paper entitled

EXAMINATION TIMETABLE GENERATION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 8, January 2025

Registration ID 171957 Research paper weblink:https://ijirt.org/Article?manuscript=171957

EDITOR EDITOR IN CHIEF

International Journal of
Innovative Research in Technology
An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

MITHALI S ANAND

In recognition of the publication of the paper entitled

EXAMINATION TIMETABLE GENERATION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 8, January 2025

Registration ID 171957 Research paper weblink:https://ijirt.org/Article?manuscript=171957

EDITOR EDITOR IN CHIEF

International Journal of
Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

TEJASHWINI B A

In recognition of the publication of the paper entitled

EXAMINATION TIMETABLE GENERATION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 8, January 2025

Registration ID 171957 Research paper weblink:https://ijirt.org/Article?manuscript=171957

EDITOR

EDITOR IN CHIEF

ISSN: 2349-6002 | ESTD Year: 2014 | Monthly Issue

International Journal of
Innovative Research in Technology (IJIRT)

An International Scholarly Open Access, Peer-Reviewed Journal

Ref No: IJIRT171957/Volume 11/Issue 8/

Subject: Publication of paper at International Journal of Innovative Research in Technology

Dear Author,

With Greetings we are informing you that your paper has been successfully published in the International Journal of Innovative Research in Technology (ISSN: 2349-6002)

Following are the details regarding the published paper.

About IJIRT: An International Scholarly Open Access Journal, Peer-Reviewed, Refereed Journal Impact factor Calculated by Google Scholar and Semantic Scholar, AI-Powered Research Tool, Multidisciplinary, Monthly, Multilanguage Journal, Indexing in All Major Databases and Metadata, Citation Generator, Impact Factor 8.01, ISSN: 2349-6002

UGC Approval	: UGC and ISSN Approved - UGC Approved
Journal No	: 47859
Link	: https://www.ugc.ac.in/journallist/subjectwisejurnallist.aspx?tid=MjM0OTUxNjI=&&did=U2VhcmNoIGJ5IEITU04=
Paper ID	: IJIRT171957
Title of the Paper	: Examination Timetable Generation
Impact Factor	: 7.367 (Calculated by Google Scholar)
Published In	: Volume 11, Issue 8
Publication Date	: 08-Jan-2025
Page No	: 1347-1351
Published URL	: https://ijirt.org/Article?manuscript=171957
Authors	: Susheeth G, Ritish N, Vidyashree B N, Mithali S Anand, Tejashwini B A, Dr Murali Parameswaran

Thank you very much for publishing your article with IJIRT. We would appreciate if you continue your support and keep sharing your knowledge by writing for our journal IJIRT.

EDITOR IN CHIEF

International Journal of Innovative Research in Technology
ISSN 2349-6002

www.ijirt.org | editor@ijirt.org | Impact Factor: 7.37 (Calculate by Google Scholar)

IJIRT.ORG Email: editor@ijirt.org

Sustainable Development Goals (SDGs)

Fig 8: SDG [12]

The "Examination Timetable Generation" project aligns with four key United Nations Sustainable Development Goals (SDGs): SDG 4 (Quality Education), SDG 9 (Industry, Innovation and Infrastructure), SDG 12 (Responsible Consumption and Production) and SDG 16 (Peace, Justice, and Strong Institutions).

SDG 4: Quality Education

The system enables efficient and conflict-free scheduling of exams, ensuring fair allocation of resources like rooms and invigilators.

Reduces errors and stress for students, enabling a smoother academic experience.

Helps educational institutions focus on improving learning outcomes rather than logistical challenges.

SDG 9: Industry, Innovation, and Infrastructure

Introduces innovation in academic infrastructure through an intelligent scheduling algorithm.

Automates processes, reducing manual labor and increasing operational efficiency in educational institutions.

Supports scalable and adaptable systems that can handle various exam scheduling complexities.

SDG 12: Responsible Consumption and Production

Reduces paper usage by providing schedules in digital formats (PDFs).

Optimizes resource utilization (e.g., classrooms, invigilators) to minimize waste.

Encourages sustainable practices in institutional operations.

SDG 16: Peace, Justice, and Strong Institutions

Facilitates transparency and fairness in scheduling, reducing conflicts in resource allocation.

Provides a systematic approach to managing institutional responsibilities, ensuring accountability.

Murali Parameswaran - ISR-03_Report_Examination Timetable Generation[1]

ORIGINALITY REPORT

15%	10%	9%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to City University Student Paper	4%
2	gentoobr.org Internet Source	1%
3	Submitted to Swinburne University of Technology Student Paper	1%
4	Submitted to Presidency University Student Paper	1%
5	Submitted to Sim University Student Paper	1%
6	wagusy.pl Internet Source	1%
7	itc.ktu.lt Internet Source	<1%
8	isteonline.in Internet Source	<1%

Submitted to Heriot-Watt University

9	Student Paper	<1 %
10	Submitted to University of Northampton Student Paper	<1 %
11	Submitted to Richard Johnson Anglican School Student Paper	<1 %
12	Submitted to University of Glasgow Student Paper	<1 %
13	Submitted to Lincoln University Student Paper	<1 %
14	Submitted to Teaching and Learning with Technology Student Paper	<1 %
15	Submitted to University of Sunderland Student Paper	<1 %
16	pastebin.com Internet Source	<1 %
17	www.un-page.org Internet Source	<1 %
18	Submitted to Aberystwyth University Student Paper	<1 %
19	Submitted to Visvesvaraya Technological University Student Paper	<1 %

20	H.L. Gururaj, Francesco Flammini, S. Srividhya, M.L. Chayadevi, Sheba Selvam. "Computer Science Engineering", CRC Press, 2024 Publication	<1 %
21	www.westcovinasabers.org Internet Source	<1 %
22	Submitted to University of Abertay Dundee Student Paper	<1 %
23	Submitted to Turun yliopisto Student Paper	<1 %
24	nextgenerations-cities.encs.concordia.ca Internet Source	<1 %
25	Submitted to Napier University Student Paper	<1 %
26	Submitted to Southern Methodist University Student Paper	<1 %
27	Submitted to Canadian International School, Singapore Student Paper	<1 %
28	huggingface.co Internet Source	<1 %
29	Ankur Gupta. "Next Generation Computing and Information Systems", CRC Press, 2024 Publication	<1 %

30	Submitted to University of Hertfordshire Student Paper	<1 %
31	Submitted to University of Portsmouth Student Paper	<1 %
32	Submitted to University of Surrey Student Paper	<1 %
33	www.dtic.mil Internet Source	<1 %
34	Submitted to Bournemouth University Student Paper	<1 %
35	home.cc.umanitoba.ca Internet Source	<1 %
36	python.hotexamples.com Internet Source	<1 %
37	www.sigmarepository.org Internet Source	<1 %
38	Chris Roda. "Essential Programming for the Technical Artist", CRC Press, 2024 Publication	<1 %
39	www.itu.int Internet Source	<1 %
40	Tatlilioglu, Eda. "Analysis of Science Curriculum and Textbooks in Terms of Sustainable Development Goals: A Case	<1 %

Study.", Middle East Technical University (Turkey), 2024
Publication

41	Wang, Zhengzheng. "Efficient Analysis and Optimization Methods for the Design of Electromagnetic Surface Unit Cells.", University of Toronto (Canada), 2024 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		