

ACKNOWLEDGMENT

We, the members of this group, sincerely express our heartfelt gratitude to Albukhary International University (AIU) for granting us the valuable opportunity to pursue our education and for providing the necessary resources and facilities that enabled us to complete this project successfully.

We would also like to extend our thanks to the School of Computing and Informatics (SCI), under the leadership of Associate Professor Dr. Basheer Riskhan, for fostering a learning environment that promotes growth and innovation.

Our deepest appreciation goes to our project supervisor, Associate Professor Ts. Dr. Leelavathi Rajamanickam, for her unwavering support, insightful feedback, and consistent guidance throughout the course of this project. Her commitment, knowledge, and mentorship have played a vital role in the successful completion of our work.

Lastly, we are truly thankful to our families for their constant encouragement, motivation, and belief in us, which has inspired us to always aim for excellence.

TABLE OF CONTENTS

DECLARATION	i
APPROVAL	ii
ACKNOWLEDGMENT	iii
TABLE OF CONTENTS	iv
LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF ABBREVIATIONS	viii
ABSTRACT	ix
CHAPTER ONE	1
INTRODUCTION	1
1.1 Introduction	1
1.2 Background	2
1.3 Problem Statement	2
1.4 Research Questions	3
1.5 Research Objectives	3
1.6 Scopes of Project	3
1.7 Significance of Project	4
CHAPTER TWO	5
LITERATURE REVIEW	5
CHAPTER THREE	12
METHODOLOGY	12

3.1	System Development Methodology	12
3.1.1	Data Description	14
3.1.2	Analysis of Data Schemas	16
3.1.3	Constraints Formulation	19
3.1.4	Hard Constraints	19
3.1.5	Soft Constraints	20
3.1.6	Flow of The Genetic Algorithm	22
3.2	System Architecture	24
3.3	Use Case Diagram	26
3.4	Activity Diagram	28
3.5	System Requirement	29
3.5.1	Functional Requirement	29
3.5.2	Non-Functional Requirement	30
3.6	Hardware and Software Requirement	31
CHAPTER FOUR		32
RESULT AND DISCUSSION		32
4.1	System Implementation	32
4.1.1	Development Process	33
4.1.2	Key Features and Functionality	35
4.1.3	User Interface (UI) and User Experience (UX)	36
4.2	Testing	41
4.2.1	Unit Testing	41
4.2.2	Integration Testing	42
4.2.3	System Testing	43
CHAPTER FIVE		45
CONCLUSION		45
5.1	Contribution To Social Business/ Social Innovation	45
5.2	Future Work	46

References (APA STYLE)	47
APPENDIX A	53
APPENDIX B	54
APPENDIX C	56

LIST OF TABLES

Table 2.1 : Comparison of Existing Approaches and Proposed Methodology	10
Table 3.1: Functional Requirement Specification	29
Table 3.1: Non-Functional Requirement Specification	30
Table 4.1: Unit Test Cases for Smart Timetable System	42
Table 4.2: Integration Test Cases for Smart Timetable System	43
Table 4.3: System Test Scenarios and Results	44

LIST OF FIGURES

Figure 3.1: Entity Related Diagram (ERD) for the timetable database	15
Figure 3.2: Genetic Algorithm flow chart	21
Figure 3.3: Chromosome representation	22
Figure 3.4 : Crossover operation of combining two parent to form an offspring	23
Figure 3.5 : Layered architecture for a smart timetable system.	24
Figure 3.6: Use case Diagram for a smart timetable system	26
Figure 3.7:Activity Diagram for a Smart Timetable System.	28
Figure 4.1: Smart Timetable System's Login Page	36
Figure 4.2: Admin Dashboard of smart timetable system	36
Figure 4.3: Timetable Tab On Admin Dashboard In Smart Timetable System	37
Figure 4.4: Algorithm Control In Tab Smart Timetable System	38
Figure 4.5: Conflict Tab In Smart Timetable System	38
Figure 4.6: User Management Tab In Smart Timetable System	39
Figure 4.7: Student Dashboard Of Smart Timetable System	39
Figure 4.8: Timetable Tab On Student Dashboard	40
Figure 4.9: Lecturer dashboard of Smart timetable System	40
Figure 4.10 : Timetable Tab On Lecturer Dashboard	41

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AIU	Albukhary International University
ANNs	Artificial Neural Networks
CPU	Central Processing Unit
DBMS	Database Management Systems
DNA	Deoxyribonucleic Acid
EAs	Evolutionary Algorithms
ERD	Entity Relationship Diagram
GA	Genetic algorithm
GAs	Genetic algorithms
HC	Hard Constraints
HOP	Head of Programme
ID	Identification
IT	Information Technology
NNs	Neural Networks
PGA	Parallel Genetic Algorithm
SC	Soft Constraints
SCI	School of Computing and Informatics
SGA	Simple Genetic Algorithm
SDGs	Sustainable Development Goals
UI	User Interface
UX	User Experience

ABSTRACT

The preparation of academic timetables is a recurring challenge for educational institutions, often resulting in scheduling conflicts, inefficient resource allocation, and a time-consuming manual process. This project introduces a Smart Timetable System designed to automate and optimize timetable generation using Genetic Algorithms (GA). By analyzing real data from the School of Computing and Informatics at Albukhary International University, the system integrates key scheduling constraints such as room capacities, lecturer availability, course distribution, and student preferences. A layered system architecture was adopted to ensure modularity, maintainability, and scalability. The Genetic Algorithm plays a central role in generating conflict-free schedules by evolving high-quality solutions across generations based on defined fitness functions. The system accommodates real-time updates, provides role-based dashboards for administrators, students, and lecturers, and supports features such as timetable export, notification of changes, and manual adjustment. Through implementation and testing, the system demonstrates a significant improvement in the efficiency, flexibility, and accuracy of timetable management, addressing the limitations of traditional scheduling approaches.

CHAPTER ONE

INTRODUCTION

1.1 Introduction

The university course timetable is an essential tool that must be prepared before the start of each semester. It involves allocating rooms and assigning lecturers to courses. Each semester, educational institutions face the challenging task of creating a timetable that meets all the specific requirements while ensuring accurate allocation of resources (Oladimeji, 2020). To prevent these issues and ensure accuracy in the timetable creation process, a smart timetable system is introduced. This system streamlines the process, reduces the time required, and avoids combinatorial problems, ensuring smooth and efficient operation.

The Smart Timetable System is a highly effective solution for scheduling problems, known for its accuracy and efficiency, often referred to as a timetable generator due to its functionality. It is one of the advanced systems designed to streamline and optimize scheduling processes, ensuring seamless operation. The growing adoption of smart systems highlights their transformative impact on efficiency and automation, with 28% of individuals using energy management tools like smart thermostats and 54% wearing devices such as fitness trackers in 2024 (Central Statistics Office, 2024). Similarly, like a skilled planner with a digital brain, the Smart Timetable System has revolutionized how schools handle their schedules. The system uses machine learning algorithms to handle all the complexities involved, making sure teachers are not double-booked, classrooms are available, and every student's schedule fits together.

Despite technological advancements, many universities still rely on manual methods to create timetables, even though computers and Artificial Intelligence (AI) have greatly enhanced the efficiency of various tasks. The development of a smart timetable system addresses the limitations of traditional methods by collecting essential data, such as classroom availability, lecturers, and courses, and utilizing AI algorithms

to generate a conflict-free schedule.

Genetic Algorithms (GAs) are implemented in the study to design efficient and suitable timetables that satisfy all the necessary requirements for a school timetable system. The system effectively eliminates class and lecture clashes, substantially reducing the time required for manual timetable creation. It is highly accurate and optimizes results by employing advanced algorithms to make the process both efficient and reliable.

1.2 Background

Smart system is an integrated system designed to gather relevant data, analyze it, and interact with its surroundings in ways that either replicate or surpass human capabilities. It achieves this by leveraging algorithms and the ability to learn and adapt over time. Examples of smart systems include smart homes, smart traffic management, smart healthcare, smart agriculture, and smart cities.

Timetabling, a scheduling problem, involves allocating activities to specific time slots and available resources while adhering to various constraints. Each activity is assigned a timeframe, and the necessary resources are allocated for their execution. Constraints that are considered when generating timetables are of two types which are hard constraints and soft constraints. **Hard constraints (SC)** are mandatory for the system to fulfill while **Soft constraints (SC)**, which are recommendable but not mandatory, non-compliance with soft constraints may still result in a functional timetable, fulfilling them contributes to a more efficient and optimized timetable (Babaagba & Arekete, 2017). **A smart timetable generator** is a system designed to provide innovative solutions to the complexities of manual method timetable creation in academic institutions. The system automates the scheduling of lectures and classes without conflicts or clashes. It features a user-friendly interface that promotes ease of use and adaptability, allowing smooth operation and clear understanding of its functionality (Barhate *et al.*, 2024).

1.3 Problem Statement

Academic institutions continue to face significant challenges in generating efficient and adaptable timetables due to complex constraints such as course conflicts, limited room capacity, and instructor availability. Traditional scheduling systems often fall short in accommodating last-minute changes and fail to provide real-time responsiveness. Andrew Reid East (2019) introduced a timetable scheduling approach that integrates Genetic Algorithm (GA) with real-time data handling, which enhances constraint resolution but lacks direct user interaction and intelligent feedback mechanisms. Similarly, Dipesh Mahajan *et al.* (2024) proposed a smart timetable system with IoT integration for push notifications and live tracking; however, the system still struggles with flexibility, scalability, and the ability to handle user-specific constraints effectively. These limitations highlight the continued need for a more robust, intelligent scheduling solution that addresses the dynamic nature of academic environments.

1.4 Research Questions

1. How can a timetable system minimize scheduling conflicts, which can lead to disruptions in academic activities and inefficient resource utilization?
2. How can a system be developed to efficiently solve the timetable generation challenges faced by colleges every academic semester?
3. How can a data-driven adaptive timetable system be developed to efficiently generate and adjust schedules based on real-time data and changing academic requirements?

1.5 Research Objectives

1. To design a timetable that minimizes scheduling conflicts among classes while ensuring the efficient allocation of resources to meet the institution's requirements.
2. To develop an efficient system for solving the timetable generation challenges faced by colleges every academic semester
3. To develop a data-driven adaptive timetable system and implement efficient algorithms that can adjust schedules dynamically.

1.6 Scopes of Project

This study is limited to the development and evaluation of a smart timetable system designed for the **School of Computing and Informatics (SCI) at Albukhary International University (AIU)**. It focuses on generating academic course timetables that take into account real constraints such as lecturer availability, student groupings, room capacity, and time slot allocation. The system uses actual timetable data from SCI and is not extended to other schools or departments within the university. The scope is confined to course-based scheduling for a single academic semester, excluding exam timetabling and multi-semester planning. Although the GA used in this system has potential applications in other scheduling contexts such as corporate event planning or public services, this study applies the technique strictly to academic course scheduling within SCI at AIU. The system also does not include comparisons with other optimization methods like simulated annealing or ant colony optimization.

1.7 Significance of Project

The novelty and contribution of the study are in designing an smart timetabling system using a GA for solving complex timetabling problems. The suggested system is more efficient and flexible compared to the conventional methods by obtaining the optimum solutions from many iterations, thus making it highly capable of meeting real-world requirements. The research is beneficial in the use of heuristic optimization methods, i.e., GA, in academic timetabling. The research has the potential to be applicable in a wide range of fields suffering from the same problems, including corporate scheduling and healthcare.

CHAPTER TWO

LITERATURE REVIEW

A timetable is perhaps the most crucial tool for efficient time management. Manual timetabling can be very challenging since a change in one timeslot may affect others or disrupt the entire schedule (Chai & Tyng, 2023). Traditional scheduling methods often rely on manual processes, which can lead to errors, conflicts, and inefficiencies. As Singh and Patel (2020) highlight, preparation of timetables not only takes considerable time but is also prone to errors. Such mistakes can lead to complications in the form of clashes in classes, overlapping schedules, and wasteful use of resources, all of which can disrupt time management for both students and lecturers. A smart timetable system effectively resolves such issues by automating the process of scheduling, thereby enabling a more precise and optimized utilization of resources like classrooms, teachers, and time slots. Such an approach minimizes the occurrence of conflicts and mistakes, thereby enhancing the overall scheduling process. Smart timetable systems are designed to alleviate human effort but increase efficiency. An intelligent system aims to create a portable, easy-to-use, efficient, and compact program that can produce high-quality timetables in less time. (Chavan *et al.*, 2024).

School timetable planning is a challenging problem that has garnered much academic attention, particularly using metaheuristic algorithms. Different studies have examined the efficacy of such methods, including the GA and Graph Coloring, and techniques that employ Database Management Systems (DBMS). The GA is particularly notable with its ability to optimize school timetable schedules for increased efficiency while handling the different constraints of the school context. GAs, based on natural selection concepts, have been successful in solving scheduling issues through optimization (Mohammed *et al.*, 2017). GAs are well-suited to handle complex constraints, such as resource utilization, conflict resolution, and student preferences, through iteratively improving solutions through mechanisms like mutation, crossover, and selection.

As noted by Firke *et al.* (2023), the construction of a new timetable app is similar to the development of a solid tool for schools that aims to surpass the existing options. The suggested app aims to simplify the process of creating timetables and, consequently, enhance speed and

efficiency. The app is made with the user in mind, thereby effortlessly avoiding issues like class clashes and temporal inefficiencies. It acts as a smart assistant for the administration of college schedules and events. Furthermore, the authors emphasized the need to protect data and provide simplicity in accessibility through additions such as the usage of time and reporting features. Over time, it is anticipated that ongoing enhancements to the application will reflect the changing needs of colleges, thereby simplifying the process of creating the timetable for all.

GAs are extensively used in the resolution of timetabling scheduling issues. GAs constitute a class of Evolutionary Algorithm (EA), which is a computational approach in computer science that emulates biological processes. Holland (1975) developed a GA, a popular category of EA, and it was founded on the mechanism of 'survival of the fittest,' as proposed by Charles Darwin. Just as Artificial Neural Networks (ANNs) are motivated by the structure of the human brain, GAs are motivated by evolutionary biology. Candidate solutions to a problem in GAs are encoded as virtual strings of Deoxyribonucleic Acid (DNA) known as chromosomes that are further subdivided into genes. Solutions are graded based on their goodness in the problem domain by a fitness function, a numerical estimate of goodness. By simulating genetic operations, such as crossover, selection, and mutation. GAs iteratively improve sets of solutions, gradually converging to optimal solutions. By doing so, GAs can search and optimize complex solution spaces without having an explicit representation of the optimal solution, but rather by a relative comparison of fitness among candidates (Andrew Reid, 2019).

GAs have wide applications in solving optimization problems since they are able to efficiently search large search spaces. GAs have applications in a very wide range of domains, such as optimization problems, machine learning and data science, healthcare and bioinformatics, robotics and control systems, games and entertainment, and industrial applications (Gen & Lin, 2023). GAs are used where the problem space is large, complex, or at least partially unknown. They build solutions piece by piece from the small building blocks of knowledge, and they often develop critical aspects of the problem along the way. This capability to work with no a priori knowledge of the domain makes genetic algorithms very versatile, for they can develop innovative and optimal solutions that were not envisioned at the design stage. Despite their reliance on randomness as an inherent part of their operation, GAs combine stochastic and guided search methods, thus allowing them to converge to solutions with impressive efficiency solutions that could be considered impracticable or unfeasible for typical

search mechanisms. The flexibility and efficiency of GAs make them essential for solving tough optimization and search problems (Mohammed et al., 2017).

According to Ogunseye and Ojuawo (2024), the basis of a GA includes several key components. First, it involves a population of hypotheses for solving a particular problem. Second, it requires a method to assess the quality and performance of each solution in the population or evaluate the "health" of the generated solutions. Third, techniques are used to combine elements of successful solutions to generate new ones that are generally better. Finally, a mutation operator is implemented to ensure diversity within the population and prevent the loss of internal variation over time.

Over time, numerous variations of GAs have been developed, including the Simple Genetic Algorithm (SGA) and the Parallel Genetic Algorithm (PGA). The key difference between them lies in how they handle populations. In PGA, the population is divided into subpopulations, with each subpopulation processed in parallel (Yua *et al.*, 2011). PGA selects individuals with the highest fitness scores from each subpopulation and migrates them to other subpopulations. Meanwhile, individuals with the lowest fitness scores are replaced by the migrated individuals. This process continues until the generation ends, ensuring weaker individuals are replaced with stronger ones, ultimately improving the overall fitness of the population. Furthermore, PGA demonstrates better performance than SGA because it can divide the population and run on different processors in parallel, whereas SGA uses a single processor, resulting in slower performance and reduced effectiveness when the population size is large. SGA, the traditional Genetic Algorithm, uses a single population, and crossover is performed on individuals with high fitness scores. Genetic Algorithms can offer more accurate solutions than conventional methods and neural networks (NNs) in certain optimization scenarios, though they often require significantly more computation time (Shorman & Pitchay, 2015).

The graph coloring algorithm is another algorithm that can be used to eradicate scheduling problems and is a useful technique for creating a timetable system. It assigns colors (representing time slots) to vertices (representing events or courses) such that no two adjacent vertices (courses with overlapping participants or requirements) share the same color. This ensures there are no conflicts, as each color corresponds to a unique time slot for a course. This algorithm models the scheduling problem as a graph, where vertices represent events (e.g.,

courses, exams) and edges represent conflicts (e.g., courses sharing the same students or resources). (Herath, 2017).

Utilization of a Database Management System (DBMS) is another method of implementing a timetable system (Chuenkrut & Achayuthakan, 2019). This method, however, is regarded as an old-fashioned method due to its tendency to be time-consuming and less effective. Although a DBMS supports data management and storage, it is not designed to render the solution of complex scheduling problems optimal, which may cause redundancy, inconsistency, as well as trouble in managing conflicts among events.

The technology behind smart timetable systems relies heavily on Artificial Intelligence and complex algorithms to meet the varying needs of educational institutions. Smart timetable systems use algorithms to consider multiple constraints, such as classroom availability, instructor schedules, student preferences and resource allocation. As Smith (2021) explains, AI enables smart timetable systems to dynamically generate timetable solutions that balance competing needs and optimize resource utilization. By continuously evaluating and adjusting the schedule, these systems can ensure that all factors, including unforeseen changes, are accommodated in real time. For example, if a teacher becomes unavailable due to illness or if a classroom is unexpectedly required for another event the smart timetable system can automatically suggest adjustments to minimize disruption.

One of the key advantages of smart timetable systems is their ability to significantly increase efficiency. As Lee (2022) points out, Smart timetable systems reduce the administrative workload by automating tasks that would otherwise require extensive manual effort. Smart timetable system allows staff to focus on more strategic initiatives rather than spending time on routine scheduling tasks. Additionally the ability of smart timetables to cater to individual preferences and constraints are another significant benefit. Singh and Patel (2020) suggest that systems allow students and faculty to input preferences such as preferred class times or specific room requirements which helps create more personalized and user-friendly scheduling experience. The level of customization enhances user satisfaction and can contribute to a more positive academic environment.

Flexibility is another vital benefit of smart timetable systems. As Williams (2021) explains, smart timetable systems can respond very rapidly to changes, like staff absence or unforeseen unavailability of rooms, an important consideration in the dynamic setting of schools.

Flexibility enables tutors and students to adapt to alterations at minimal disruption, thus enabling the academic schedule to progress. Moreover, smart timetabling systems also possess the capability to cater for last-minute changes or special requests, further personalizing the overall experience to better fit the institution's requirements

However, despite their clear advantages there are several challenges associated with implementing a smart timetable system. One of the primary challenges is the initial setup and integration with existing scheduling systems. According to Roberts and Harris(2020) the customization and technical support required during the initial implementation phase can be costly and time consuming. Institutions may have to invest in new infrastructure or make major adjustments to their current Information Technology (IT) infrastructure in order for the smart timetable system to run smoothly. Complexity in handling smart timetable systems sometimes results in resistance from staff and students towards new technology that may hinder the adoption process.

Brown and Kumar (2021) note that artificial intelligence algorithms have no guarantee of catching the nuances of human preference or last-minute changes, and which can have suboptimal scheduling outcomes. For instance, artificial intelligence can miss the unique scheduling requirements of special needs students, or not always schedule for optimal use of time. Although such programs possess immense capabilities, they are not perfect, and there might still be a need for some human touch for the schedule to fit everyone's needs.

Table 2.1: Comparison of Existing Approaches and Proposed Methodology

Aspect	Achini Kumari Herath (2017)	Mohammed et al.,(2017)	Andrew Reid East (2019)	Dipesh Mahajan et al. (2024)	Proposed Project
purpose	Genetic Algorithm For University Course Timetabling Problem	Genetic Algorithm for Exam Timetabling	Timetable Scheduling via Genetic Algorithm	Timely Trigger: A Smart Timetable Generator	A smart timetable system for school
Algorithm used	Genetic Algorithm (GA), graph coloring, mathematical programming algorithms, Tabu search, constraint with custom fitness functions. DBMS	Genetic Algorithm (GA).	Genetic Algorithm (GA) integrated with cloud computing	Genetic Algorithm integrated with IoT for push notifications and live tracking	Genetic Algorithm with penalty-based fitness functions and simple mutation operations tailored for educational scheduling
Key features	Explores encoding, mutation, and fitness evaluation for timetable optimization.	Emphasizes the ability of GA to minimize scheduling conflicts.	Combines GA with real-time data exchange for resolving constraints dynamically.	Push notifications, real-time tracking, error detection, and Excel integration for data management.	Focuses on predefined constraints like room capacity, and time slots, with automated conflict detection.

Improvements	Suggests improving population diversity and fitness functions	Suggests improving fitness functions for better optimization.	Enhancing the scalability of algorithms for larger datasets.	Plans to integrate IoT for precise tracking and administrative dashboards.	Proposes optimization of algorithm speed by reducing computational complexity and adding features like manual adjustment of generated timetables
Future Work	Exploring hybrid approaches to improve solutions for larger datasets.	Exploring hybrid algorithms with GA and other evolutionary methods.	Incorporating machine learning models for predictive timetable generation.	Incorporating IoT for detailed analytics and improving UI/UX for users.	Adding support for constraint customization by users (e.g., prioritizing specific courses or time slots) and improving scheduling accuracy.
Strengths	Clear encoding and detailed genetic operations for timetabling.	Provides a strong basis for GA application in timetabling	Demonstrates modern application of GA with cloud tools	Combines scheduling with modern features like notifications and tracking.	Simple, efficient design focusing on ease of implementation, practical constraints, and a user-friendly interface.

CHAPTER THREE

METHODOLOGY

3.1 System Development Methodology

The development of the Smart Timetable System followed the Waterfall model, which is a linear and sequential system development methodology. This model was well-suited for the project because it allowed for a structured and orderly progression, where each phase was completed before moving on to the next. The approach ensured that all requirements were clearly defined upfront, and it provided a solid framework for thorough documentation and detailed planning at each stage of development. By adhering to this methodology, the team was able to maintain focus and clarity throughout the process, minimizing the risk of errors and ensuring that the final product met the school's specific needs.

The system development process comprises the following phases:

- 1. Requirement Elicitation:** To ensure the system addresses real-world academic scheduling needs, **requirement elicitation** was conducted through **interviews with the Dean of SCI**. This helped identify key user expectations, operational constraints, and the major challenges faced with the current manual timetable process. Both functional and non-functional requirements were gathered during this phase. Functional requirements included features like conflict-free timetable generation, room and teacher availability management, and user-specific views (e.g., for students and lecturers). Non-functional requirements included usability, reliability, and system response time.
- 2. Requirement Analysis:** The requirements obtained were analyzed to classify and prioritize them based on criticality. This phase involved identifying constraints such as maximum student group sizes, lecturer teaching hours, course dependencies, and limited room resources. The constraints were categorized into **hard constraints** (which must not be violated) and **soft constraints** (which can be optimized). The analysis also involved creating use case models to visualize system interactions and data flow.
- 3. System Design :** The system architecture was designed using a layered architecture to ensure maintainability, scalability, and separation of concerns. It consists of five layers.

The **User Interface Layer** handles login and registration, input forms, and timetable viewing for students, lecturers, and administrators. The **Application Layer** manages core operations such as timetable generation, notifications, and constraint validation. The **Data Layer** is responsible for managing the database, which includes information on users, courses, rooms, timetables, and constraints. The **Integration Layer** incorporates the Genetic Algorithm (GA) module, along with data transformation processes and cross-layer communication mechanisms. **Infrastructure Services Layer** provides support for essential backend services such as security, backup and recovery, system integration, and scalability. To aid system development, Entity Relationship Diagrams (ERD) and data dictionaries were created to represent the data structure and system logic. The GA was implemented to process various constraints as input and generate high-quality, conflict-free timetable solutions through iterative evolutionary processes.

4. **System Implementation:** The system was implemented using appropriate development tools and technologies. Python for the GA logic, a web framework for the interface, and MySQL for databases. The Genetic Algorithm was programmed to handle constraint satisfaction by assigning fitness scores to candidate timetables and applying crossover and mutation operators to evolve better solutions over generations.
5. **System Testing:** Once the implementation was completed, the system underwent several levels of testing to ensure its functionality and reliability. Unit testing was conducted on individual modules such as input validation and the fitness function to confirm that each component performed as expected. Integration testing followed to verify that the different modules, including the database and the Genetic Algorithm (GA) engine, interacted seamlessly and exchanged data correctly. Finally, system testing was carried out using actual timetable data from the School of Computing and Informatics (SCI) to assess the overall performance of the system in terms of accuracy, efficiency, and its ability to satisfy all predefined constraints.

3.1.1 Data Description

Data for this research were collected from the SCI, whose two programmes are Computer Science and Data Science. The two programmes have a blend of common and programme-specific courses and operate under one building and centralized faculty. Both the two programmes have a Head of Programme (HOP) whereas general management is headed by a Dean. The college contains around 300 students, who are studying various graduate and undergraduate degrees. The campus comprises lecture theaters, laboratories, and lecture halls, which cater to the theoretical as well as the practical aspects of the curriculum.

The information also includes lecturers' details, class timetables, and available resources such as rooms and equipment. Lecturers' availability, class timings, and room availability are important factors in the scheduling problem to construct the timetable. The objective of the intelligent timetable system is to make the scheduling process easier and efficient, thus making the utilization of resources effective and reducing conflicts. The system needs to consider multiple variables such as overlapping courses, room allocation, and instructor availability to create an interference-free schedule for both students and instructors.

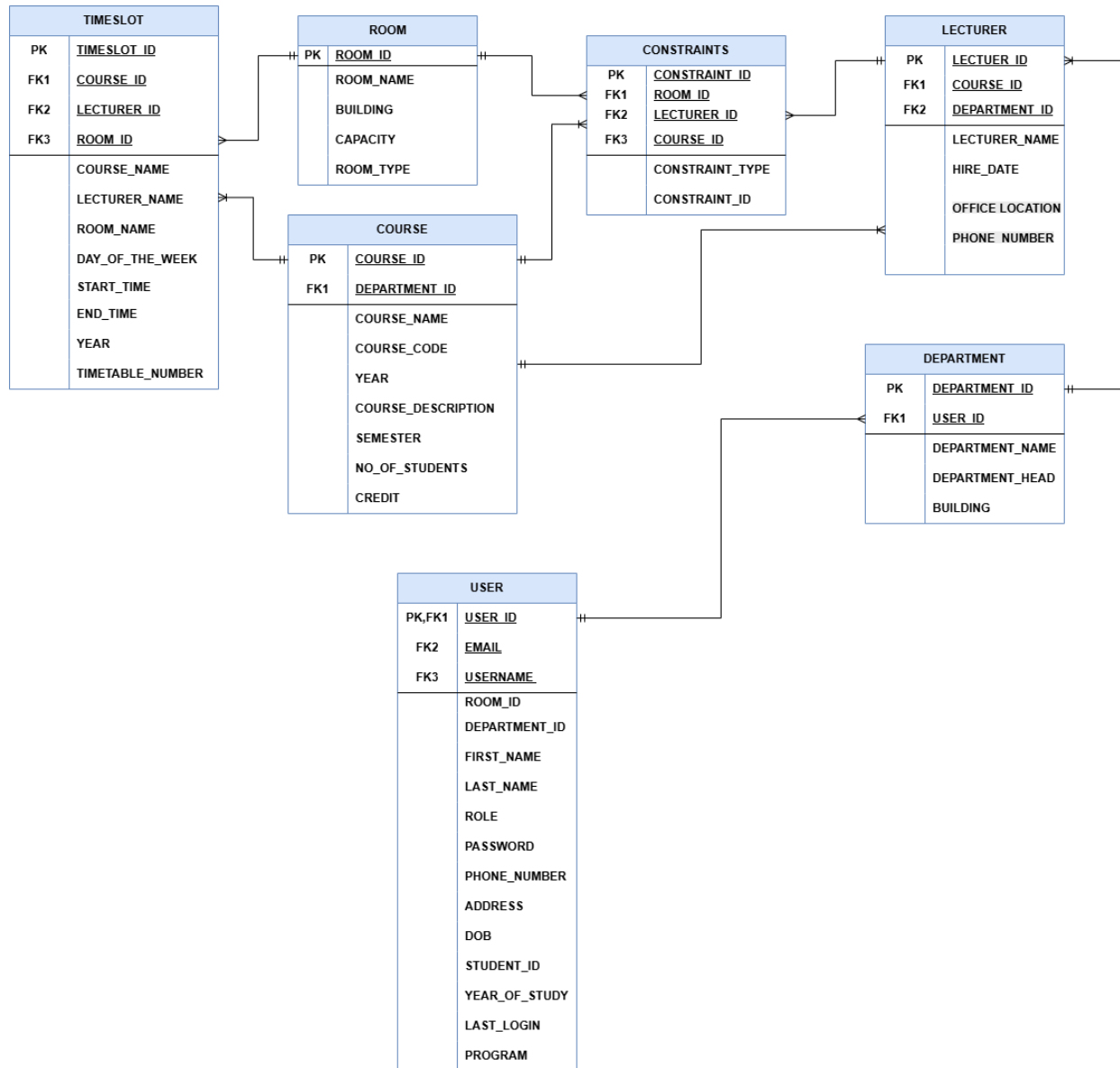


Figure 3.1: Entity Related Diagram (ERD) for the timetable database.

3.1.2 Analysis of Data Schemas

Figure 3.1 presents the data schema used in the Smart Timetable System, which organizes and manages data essential for academic scheduling. The system processes input data and classifies it into the following relational schemas:

1. **Course Schema:** This schema is used to store all the relevant details of a course in a manner so as to be retrieved easily. The schema has the following:
 - **COURSE_ID:** Unique identifier for the course.
 - **COURSE_NAME:** The full title of the course (e.g., “Introduction to Programming”).
 - **COURSE_CODE:** Short code used to identify the course (e.g., CSC101).
 - **COURSE_DESCRIPTION:** A brief overview of what the course is about.
 - **DEPARTMENT:** Indicates which department offers the course.
 - **SEMESTER:** Shows in which semester the course is offered (e.g., Semester 1).
 - **NO_OF_STUDENTS:** Indicates the number of students registered for the course useful for room assignment.
 - **CREDIT:** Shows how many credit hours the course is worth.
2. **Room Schema:** The schema relating to lecture rooms, which should encompass a feature that will ensure its proper organization and identification. These selections include
 - **ROOM_ID:** Unique ID that identifies each room.
 - **ROOM_NAME:** The name or label given to the room (e.g., "Lab 3", "Auditorium A").
 - **BUILDING:** The building in which the room is located.
 - **CAPACITY:** The maximum number of people the room can accommodate.
 - **ROOM_TYPE:** The purpose of the room, such as "Lecture Hall", "Laboratory", or "Office".
3. **Time Slot Schema:** This schema is employed in packing time-related information. Other schemas apply it. It contains the following:
 - **TIMESLOT_ID:** Unique identifier for each scheduled session.
 - **COURSE_ID:** Indicates which course is being held during the time slot.
 - **ROOM_ID:** Shows where (in which room) the session will be held.

- **LECTURER_ID:** Indicates who will be teaching the session.
 - **DAY_OF_THE_WEEK:** Specifies the day (e.g., Monday, Tuesday).
 - **START_TIME:** The time the session begins.
 - **END_TIME:** The time the session ends.
 - **SEMESTER:** Indicates the academic semester.
 - **YEAR:** Specifies the academic year (e.g., 2025).
 - **TIMETABLE_NUMBER:** A grouping or tracking number that can identify specific schedules (e.g., a batch timetable).
 - **ROOM_NAME:** A repeated attribute for easy viewing/reporting (redundant, since linked via ROOM_ID).
 - **COURSE_NAME:** The course title for easier reference in schedules (also redundant).
 - **LECTURER_NAME:** The name of the lecturer for viewing timetables easily (redundant too).
4. **Lecturer Schema:** The schema holds all information concerning lecturers, whereby easy retrieval of their details can be abstracted from this schema. They entail:
- **LECTURER_ID:** Unique identifier for each lecturer.
 - **COURSE_ID:** Links the lecturer to a specific course they are responsible for.
 - **LECTURER_NAME:** The full name of the lecturer.
 - **DEPARTMENT:** Indicates the department the lecturer belongs to.
 - **HIRE_DATE:** The date the lecturer was hired..
 - **OFFICE_LOCATION:** Indicates where the lecturer's office is located.
 - **PHONE_NUMBER:** The lecturer's contact number.
5. **Department Schema:** This contains the descriptions of any department that belongs to the institution. It has:
- **LECTURER_ID:** Unique identifier for each lecturer.
 - **COURSE_ID:** Links the lecturer to a specific course they are responsible for.
 - **LECTURER_NAME:** The full name of the lecturer.
 - **DEPARTMENT:** Indicates the department the lecturer belongs to.
 - **HIRE_DATE:** The date the lecturer was hired.
 - **OFFICE_LOCATION:** Indicates where the lecturer's office is located.

- **PHONE_NUMBER:** The lecturer's contact number.
6. **User Schema:** This is a Schema that holds the details of the system stakeholders, i.e., lecturers, students, and administrators. It has the following:
- **USER_ID:** A unique identifier assigned to every user.
 - **DEPARTMENT_ID:** Indicates which academic department the user is associated with (e.g., Computer Science).
 - **ROOM_ID:** Refers to a room assigned to the user (typically for lecturers or staff like an office).
 - **FIRST_NAME:** The user's given name.
 - **LAST_NAME:** The user's family name.
 - **EMAIL:** The user's institutional or personal email address.
 - **ROLE:** Defines the user's function in the system (e.g., "Student", "Lecturer", "Admin").
 - **PASSWORD:** The user's login password (ideally stored in hashed form for security).
 - **PHONE_NUMBER:** Contact number for the user.
 - **ADDRESS:** The home or mailing address of the user.
 - **USERNAME:** The name used to log into the system.
 - **DOB:** The user's date of birth.
 - **STUDENT_ID:** A unique student registration number (relevant only for students).
 - **PROGRAM:** The academic program the student is enrolled in (e.g., BSc Information Technology).
 - **YEAR_OF_STUDY:** The current academic year the student is in (e.g., Year 1, Year 2).
 - **LAST_LOGIN:** Date and time when the user last logged into the system.
7. **Constraints Schema:** Captures restrictions and limitations that guide how the schedule should be generated.
- **CONSTRAINT_ID:** Unique identifier for the constraint.
 - **COURSE_ID:** Identifies which course the constraint affects.
 - **LECTURER_ID:** Identifies which lecturer the constraint is associated with.

- **ROOM_ID:** Identifies which room is subject to the constraint.
- **CONSTRAINT_VALUE:** Describes the constraint itself (e.g., “Unavailable Monday 8–10am”).
- **CONSTRAINT_TYPE:** Classifies the nature of the constraint — it could be time-based, capacity-related, or based on availability.

3.1.3 Constraints Formulation

In developing an efficient and effective timetable system with GA, one should take into account some restrictions. Constraints are categorized as **hard constraints**, or obligatory and should be satisfied in order for the timetable to be valid, and **soft constraints**, or not obligatory but preferable because they improve the general quality and usability of the timetable. Both types of constraints are essential in determining the system's ability to meet mandatory requirements and also optimize for other wants. Below is a detailed overview of the hard and soft constraints derived from the data.

3.1.4 Hard Constraints

1. No student should attend more than one lecture at the same time. (HC1)
2. A lecturer cannot teach more than one course at the same time. (HC2)
3. No room should host more than one lecture at the same time.(HC3)
4. Fridays from 12:30–14:30 pm are reserved for Muslim prayers.(HC4)
5. Room capacity must not be exceeded by the number of enrolled students. (HC3)
6. Courses must only be assigned to their designated lecturers. (HC4)
7. All listed courses must appear in the final timetable. (HC5)
8. Courses must fulfill the minimum required number of sessions. (HC6)
9. All scheduled classes must take place between 08:30 and 18:30 only. (HC7)

3.1.5 Soft Constraints

1. Early morning classes between 08:30 and 10:00 should be avoided. (SC1)
2. Avoid continuous lectures or blocks of the same course on the same day; it is preferable to spread them throughout the week. (SC2)
3. Evening lectures (18:00–20:00) should be assigned to rooms with air conditioners to ensure uninterrupted sessions and maintain comfort. (SC3)
4. The number of students attending a course should not exceed the number of seats available in the lecture room. (SC4)
5. Avoid weekend's classes for students. (SC5)
6. Courses labeled A, B, C, or D should be scheduled in lab rooms. (SC6)
7. Weekend classes should be minimized or avoided when possible. (SC7)
8. Lecturers should be given sufficient rest time between consecutive classes. (SC8)
9. Evening classes should be assigned to rooms equipped with air conditioning. (SC9)
10. Late evening classes between 16:00 and 18:30 should be avoided. (SC2)

3.1.6 Flow of The Genetic Algorithm

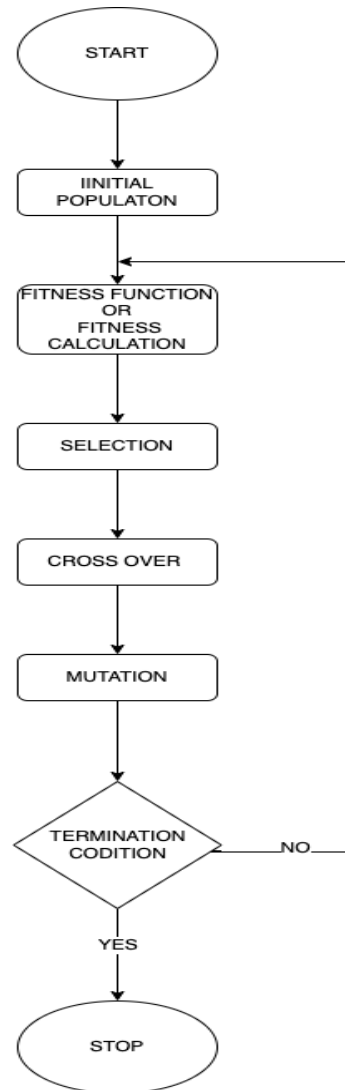


Figure 3.2: Flow Chart of Genetic Algorithm

Figure 3.2 shows the flow chart of a **Genetic Algorithm (GA)**, and it illustrates the main stages involved in evolving a population of potential solutions towards optimality. Each of the components in the flow chart is explained below:

1. **Initial Population of Chromosomes:** The process begins by generating an initial population of chromosomes, where **each chromosome represents a complete timetable** and each **gene within it refers to one class session** including subject, time, room, and lecturer assignments. These chromosomes are generated randomly at first but must meet

basic hard constraints, such as valid lecturer-course-room combinations. The size of this population directly affects the algorithm's ability to explore diverse scheduling patterns. a small population may limit the algorithm's ability to explore diverse timetable solutions, while a large one may slow down convergence (Obaid et al., 2015; Park & Kim, 2017). **Figure 3.3** illustrates how genes form chromosomes and how multiple chromosomes make up the full population. Therefore, selecting an appropriate size depends on institutional needs and computational resources. By distributing feasible solutions across the search space, the random initialization avoids premature convergence and enables broad exploration from the outset.



Figure 3.3: Chromosome representation (each gene represents a class session)

2. **Fitness Function:** The fitness function evaluates how well a timetable satisfies both hard and soft constraints. Hard constraints might include avoiding overlapping sessions for lecturers or rooms, while soft constraints may involve preferences like minimizing gaps between classes or assigning morning slots to preferred lecturers. The function quantifies each solution's quality by assigning a score typically a positive integer that reflects how well it meets these goals (Herath, 2017). A properly constructed fitness function directly guides selection, crossover, and mutation processes by favoring solutions with fewer violations and better alignment with institutional scheduling policies. For example, a schedule minimizing conflicts and maximizing room utilization will rank higher. Incorporating domain knowledge, such as departmental rules and course types, further ensures relevant and practical timetables (Abbaszadeh & Saeedvand, 2014).
3. **Selection:** In timetable generation, selection determines which timetable solutions (chromosomes) are allowed to produce offspring. The goal is to retain high-quality schedules while discarding unfit ones. Techniques like roulette wheel selection, which picks candidates based on fitness probability, or tournament selection, which chooses

from randomly selected subsets, are commonly used (Gen & Lin, 2023). Rank selection and elitism help prevent premature convergence by preserving diverse and competitive solutions. Effective selection increases the chances of producing better timetables in each generation, avoiding repetition and ensuring that only the fittest combinations of timeslots, courses, and resources are propagated.

4. **Crossover:** Crossover generates new timetable solutions by recombining parts of two selected parent schedules. For instance, one part of the offspring may inherit morning sessions from one parent and afternoon sessions from the other. This mixing can be implemented using single-point, two-point, or uniform crossover techniques. However, care must be taken as breaking apart or combining incompatible segments can destroy beneficial structures in the timetable. Random crossover points and careful selection strategies help in maintaining a good balance between exploration and preservation of good features. The crossover rate must be optimized: frequent crossovers speed up convergence, but excessive crossover can flood the population with unstable or poor-quality solutions. **Figure 3.4** shows how this recombination works, exchanging genes (class sessions) between two parent timetables.

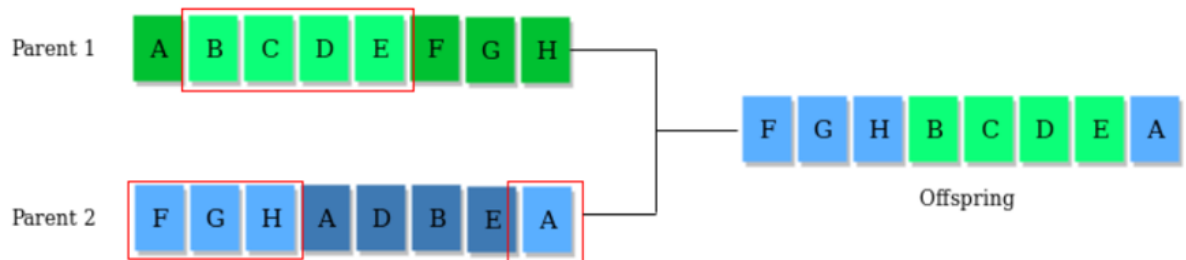


Figure 3.4 : Crossover operation of combining two parent to form an offspring

5. **Mutation:** Mutation introduces random variations into schedules to maintain diversity and prevent the algorithm from getting stuck at local optima. For timetable problems, this could involve swapping two timeslots, changing a course assignment, or moving a session to a new room. The mutation must preserve timetable validity for instance, it must not assign the same room to two classes at the same time. Methods like swapping or shuffling are preferred over bit-flipping to preserve the uniqueness and feasibility of course-room-time combinations (Limkar et al., 2015). The mutation rate should be kept

low to prevent excessive disruption of near-optimal schedules, but high enough to encourage exploration and escape poor solutions.

6. **Termination:** The algorithm stops when one of several conditions is met: a maximum number of generations is reached, a solution achieves a target fitness level, or no significant improvement occurs over successive generations. For timetable scheduling, this ensures that the final timetable is both efficient and practical. Using hybrid termination criteria e.g., combining a fitness threshold with generation limits ensures a balance between search quality and computational efficiency. Proper termination avoids excessive computation and provides a timely output of an optimized class schedule that satisfies institutional requirements.

3.2 System Architecture

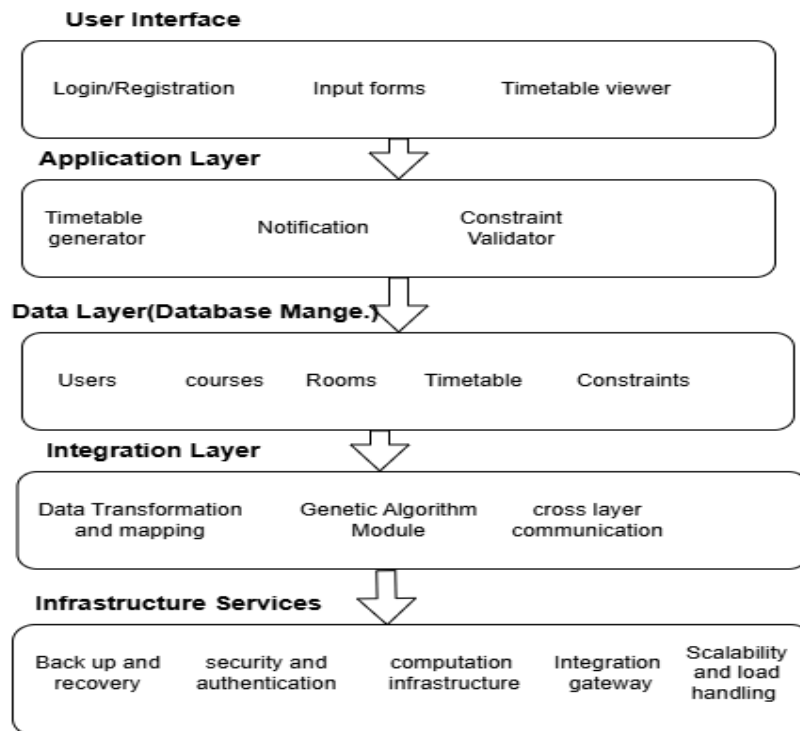


Figure 3.5 : Layered architecture for a smart timetable system.

Figure 3.5 illustrates the use of a layered architecture for the smart timetable system. This architectural style was selected because it promotes maintainability, scalability, and a clear division of responsibilities. By assigning distinct functions to each layer, the system becomes more modular, making it easier to test, debug, and implement future upgrades efficiently. Each of the layer in the architecture are explained below

- **User Interface Layer:** Allows users to interact with the system and input data like preferences and constraints.
- **Application Layer:** Provides core services such as timetable generation, conflict resolution, and notifications.
- **Data Layer:** Stores all system data, including rooms, lecturers, courses, and constraints.
- **Integration Layer:** Processes collected data, applying algorithms like the Genetic Algorithm to generate optimized timetables.
- **Infrastructure Services Layer:** Offers essential support features such as security, backups, and recovery to ensure system reliability.

3.3. Use Case Diagram

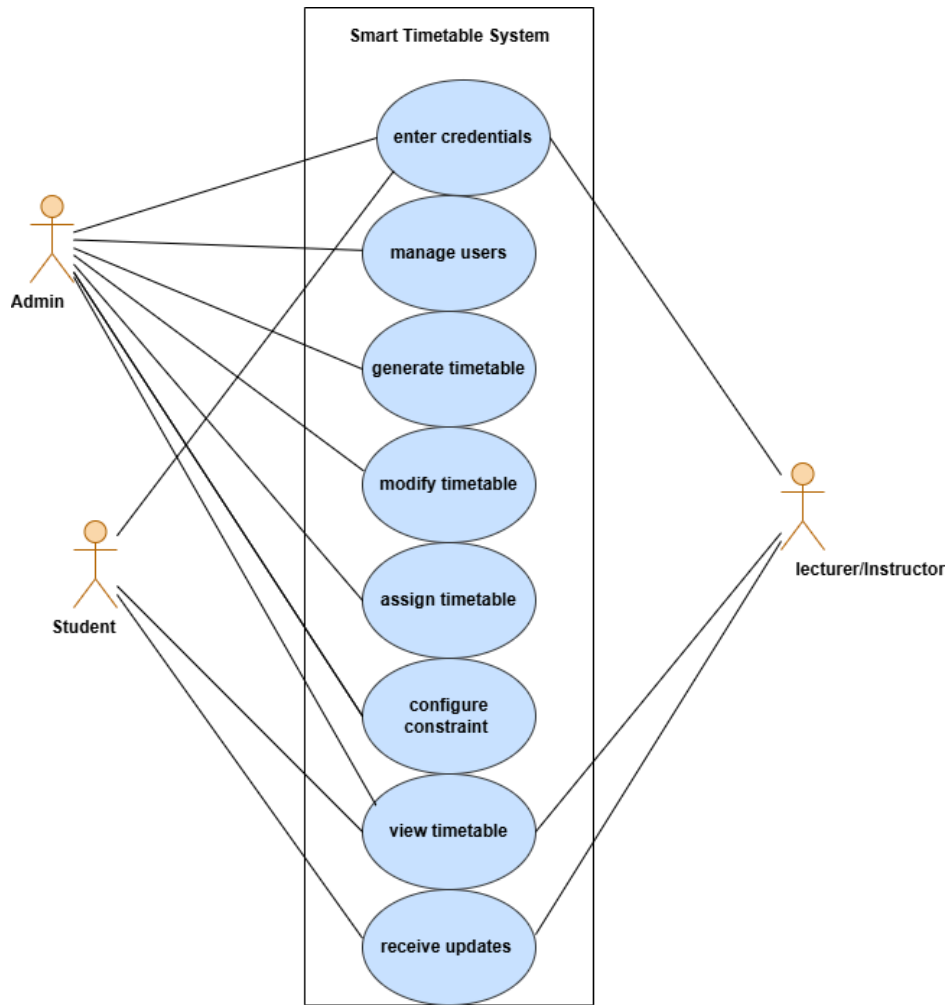


Figure 3.6: Use case Diagram for a smart timetable system

Figure 3.6 shows The use case diagram and illustrates the interaction between the actors and the system's functional requirements. As depicted in the diagram, there are three actors: the admin, the student, and the lecturer/instructor. The admin has overall control of the system, while the lecturer and the students serve as the primary stakeholders of the system.

The Smart Timetable System is composed of several core modules, each aligned with the specific responsibilities of different user roles as shown in the Use Case Diagram (Figure 3.6). These modules correspond to the main system interactions such as logging

in, viewing timetables, generating schedules, and managing users and are structured to ensure efficient and secure use for administrators, lecturers, and students.

1. **Admin Module:** The admin module has the most comprehensive control over the system, corresponding directly to several use cases in the diagram including login, manage users, generate timetable, modify timetable, assign timetable, and configure constraint. Admins can log into the system and manage all users adding, updating, or removing students, lecturers, or fellow admins. They also initiate the generation of class timetables using the genetic algorithm, make manual adjustments when necessary, and assign finalized schedules to students and lecturers.
2. **Student Module:** The student module is focused on usability and access, linked directly to the login, view timetable, and receive updates use cases. Once logged in, students can view their personalized class schedules, including subjects, instructors, rooms, and time slots. If any changes are made to the schedule such as room reassignments or time shifts students automatically receive updates, ensuring they stay informed without needing to make inquiries.
3. **Lecturer Module:** For lecturers and instructors, the module provides access to schedules and course assignments, associated with the login, view timetable, and receive updates use cases in the diagram. After logging in, lecturers can see the classes they are assigned to, the locations, and timings. The system ensures that any updates made by the admin (e.g., conflict resolution or reassignment) are reflected immediately on the lecturer's dashboard, helping them prepare effectively for each class.
4. **Shared Components :** Certain functionalities are shared among all user types. The most important of these is the login feature, which is universal for admins, students, and lecturers. Once authenticated, the system determines the user role and grants access to the appropriate module and features. Another shared component is to receive updates, which keeps all users notified when there are changes in schedules or assignments.

3.4 Activity Diagram

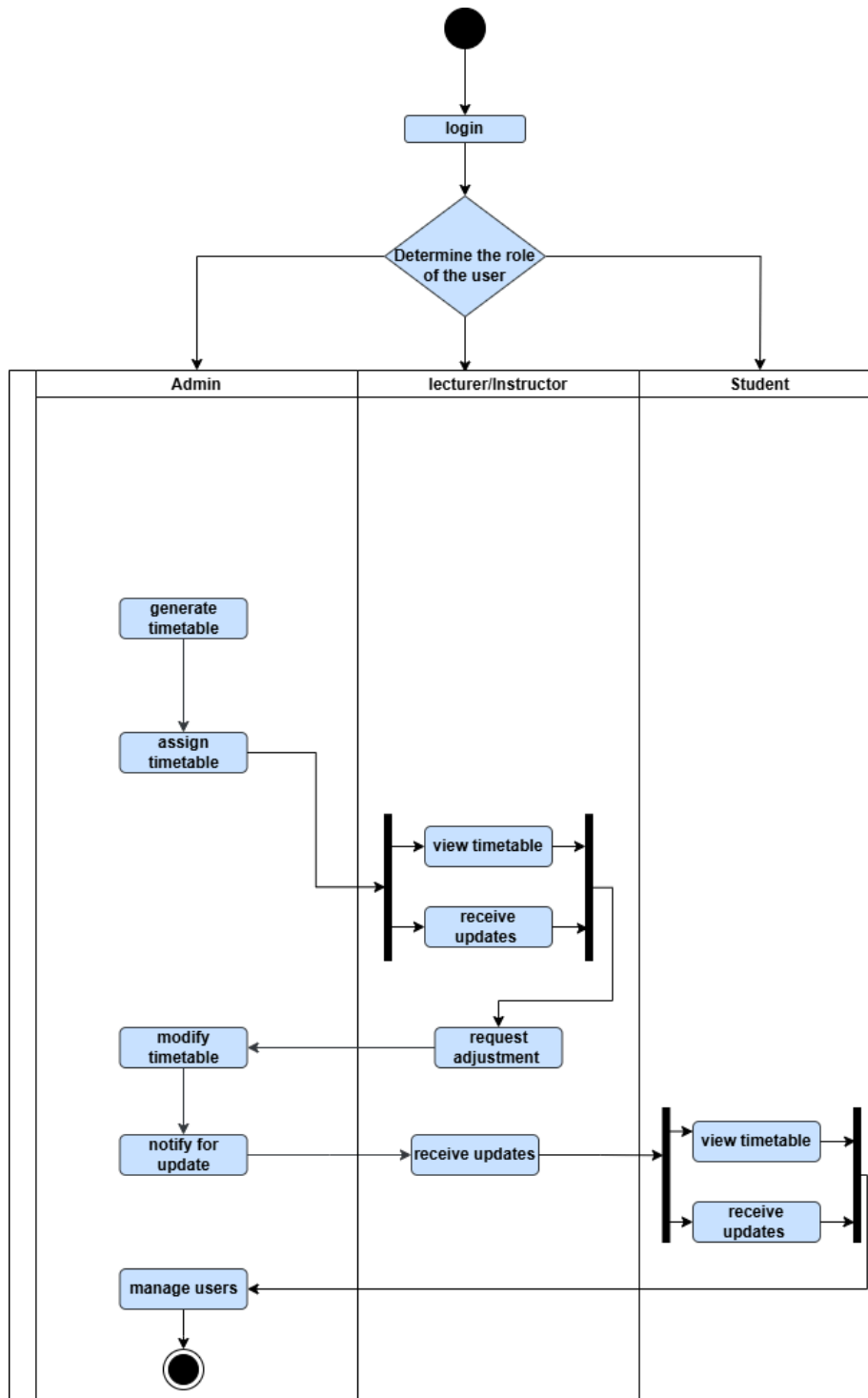


Figure 3.7: Activity Diagram for a Smart Timetable System.

Figure 3.7 illustrates the activity flow of the Smart Timetable System, showing how different user roles (Admin, Lecturer/Instructor, and Student) interact with the system after logging in. The process begins with all users logging in and the system identifying their role. Admins can generate and assign timetables, modify them if necessary, manage users, and send updates. Lecturers are allowed to view their timetables and request adjustments. Once changes are made, they receive the updated version. Students can view their personal class schedules and receive automatic updates whenever any modifications are made.

3.5 System Requirement

3.5.1 Functional Requirements

3.1: Functional Requirement Specification

No.	Requirement	Description
FR1	User Management	Admin can retrieve and manage user data securely, ensuring no unauthorized access or data leakage.
FR2	Timetable Generation	Admin generates the timetable using a genetic algorithm that considers constraints provided by the users.
FR3	Modification of Timetable	Admin can modify generated timetables based on lecturers' feedback or complaints.
FR4	Assignment of Timetable	Timetables are assigned to specific faculties or schools accurately and without conflict.
FR5	Configuration Of Constraints	Constraints entered by users are applied during timetable generation to fulfill specific needs.

FR6	User Authentication	Users must sign in or sign up to access the system securely.
FR7	Request for Adjustment	Lecturers can request changes to the timetable if it does not match their available time slots.
FR8	View Timetable	Lecturers and students can view their personalized timetables after login.
FR9	Receiving Updates	If any changes occur, the system notifies lecturers and students with real-time updates.

3.5.2 Non-Functional Requirements

3.2: Non-Functional Requirement Specification

No.	Category	Requirement Description
NFR1	Performance	The system should generate a timetable within 5 minutes for a medium-sized university, ensuring efficiency even under heavy processing.
NFR2	Usability	The system should provide an intuitive and user-friendly interface for all users—admins, lecturers, and students—with minimal learning effort.
NFR3	Security	Login credentials and timetable data must be secured using encryption and access control mechanisms to prevent unauthorized access or misuse.

NFR4	Accessibility	The system should be accessible to users with disabilities, supporting keyboard navigation, screen readers, and other accessibility tools.
------	---------------	--

3.6 HARDWARE AND SOFTWARE REQUIREMENTS

To design and implement an effective smart timetable system for schools, appropriate hardware and software resources must be in place to support the system's functionality and performance. The hardware requirements include a modern computer system with sufficient processing power, memory capacity, storage, and reliable internet access. Standard input devices such as a keyboard and mouse are also essential for system interaction. These components provide the necessary foundation for smooth development, deployment, and usage of the timetable system.

On the software side, the system can be developed using widely supported operating systems like Windows, macOS, or Linux. Development environments such as IntelliJ IDEA or Eclipse, alongside database management systems like MySQL or SQLite, support application development and data storage. To enhance development efficiency, frameworks such as Spring Boot or Django can be used, while version control tools like Git help in managing code changes. Testing tools ensure system reliability, and proper documentation tools facilitate ongoing maintenance. Collectively, these software tools and platforms provide a suitable environment for building and maintaining the smart timetable system.

CHAPTER FOUR

RESULT AND DISCUSSION

4.1 System Implementation

The Smart Timetable System is designed to automate and optimize the process of creating and managing academic timetables for educational institutions. The primary goal of this system is to eliminate the inefficiencies and errors associated with manual timetable creation, ensuring that all scheduling constraints are met while optimizing resource utilization. The system leverages advanced algorithms, specifically GA, to generate conflict-free timetables that are both efficient and flexible.

Components of the System

Figure 3.5 illustrates the layered architecture of the Smart Timetable System, which is composed of five main components that work together to ensure the system's efficiency, scalability, and usability.

1. **User Interface Layer:** This layer's goal is to make sure that every user can communicate with the system without any problems. Through user authentication, it offers secure access, limiting data entry and modification to authorised personnel only. Data input forms enable administrators to enter essential information such as course details, lecturer availability, and room capacities. Students and instructors may see their schedules with ease thanks to the timetable viewing feature, and everyone is always aware of the most recent information thanks to the real-time notifications system that alerts users of any changes or updates.
2. **Application Layer:** The Smart Timetable System's fundamental functions are managed by the Application Layer. Using the GA, it automatically generates optimal schedules while making sure that all predetermined constraints are fulfilled. To ensure a schedule free of conflicts, the conflict resolution module finds and fixes any scheduling problems. By giving administrators the ability to manually alter the schedule as necessary, modification tools offer flexibility. In order to keep all parties aware and able to adjust as necessary,

the notification service also provides users with real-time updates regarding any modifications to the schedule.

3. **Data Layer (DBMS):** This layer makes sure that all of the information needed to create a schedule is safely saved and easily accessible. Important data, including user identification, course information, room capacity, and constraints, are stored in the DBMS. Structured data schemas arrange this data so that the Application Layer can process and retrieve it quickly. In order to ensure data integrity and enable the Application Layer to retrieve and change data as needed, the Data Access Layer offers safe and effective database access..
4. **Integration Layer:** This layer acts as a link between the application layer and the data layer. It creates optimal timetables that satisfy all given requirements by processing the gathered data and using the GA. In order to facilitate effective timetable production, this layer makes sure that data is appropriately formatted and processed for the GA. To ensure smooth integration and operation, the Integration Layer forwards the optimised timetable to the Application Layer for additional processing and user interaction.
5. **Infrastructure Services Layer:** This layer guarantees the Smart Timetable System's general dependability and security. By preventing illegal access and data breaches, the Security Module safeguards user data and guarantees safe system access. The System Monitoring component continuously monitors system performance, identifying potential issues before they impact users and ensuring optimal operation.

4.1.1 Development Process

The development of the Smart Timetable System was carried out in five structured phases, following an incremental approach to system building while integrating a GA for timetable optimization.

Phase 1: Project Setup and Requirements Refinement, the development environment was configured, including integrated development environments (IDEs), version control, and the overall project structure. System requirements were finalized in consultation with stakeholders, identifying key entities such as teachers, students, subjects, and rooms. The team then planned API endpoints for functionalities like user management, timetable creation, and GA operations.

Additionally, the GA's internal structure was outlined by defining chromosome representation and the handling of constraints.

Phase 2: Database and Backend Foundation, relational tables were created for all relevant entities including users, courses, rooms, time slots, and constraints. Standard CRUD (Create, Read, Update, Delete) operations were implemented to manage this data effectively. User authentication was secured using JWT (JSON Web Token), and the necessary API endpoints were built to support both data handling and algorithm execution.

Phase 3: GA Implementation involved designing the chromosome structure for timetable generation, encoding elements like classes, lecturers, rooms, and time slots. A fitness function was created to evaluate how well each timetable met both hard and soft constraints. Key genetic operators such as selection, crossover, and mutation were implemented, along with repair mechanisms to address any invalid schedules. The algorithm was fine-tuned for performance and solution diversity.

Phase 4: Frontend Development, role-based dashboards were developed for administrators, lecturers, and students. The admin dashboard included full management features for assigning rooms, generating timetables, and resolving scheduling conflicts. Lecturers could access their personalized teaching schedules, while students could view their own class timetables with real-time updates. Data input forms were added for managing lecturers, courses, and constraints. Visual components were designed to clearly present timetables, and a dedicated interface was built to identify and resolve conflicts efficiently.

Phase 5: Integration and Testing ensured full connectivity between frontend and backend systems, allowing smooth communication and data flow. Both unit and integration testing were conducted to validate CRUD operations, the genetic algorithm's behavior, and overall system functionality, ensuring the Smart Timetable System performed reliably and as intended.

4.1.2 Key Features and Functionality

The Smart Timetable System incorporates several core features and functionalities that address the challenges associated with manual timetable creation and management. These features are designed to ensure the system meets the needs of educational institutions while providing a user-friendly and efficient solution.

1. User Management:

- Admin Control: The admin has overall control of the system, managing user information, and ensuring data security.
- User Authentication: Users can access the system through secure login credentials, ensuring that only authorized personnel can make changes to the timetable.

2. Timetable Generation:

- Automated Scheduling: The system uses a GA to generate optimized timetables that meet all specified constraints.
- Conflict Resolution: The algorithm ensures that no student or lecturer is scheduled for multiple activities at the same time and that room capacities are not exceeded.

3. Modification and Adjustment:

- Manual Adjustments: The admin can manually modify the timetable if needed, allowing for flexibility in response to unexpected changes.
- Request for Adjustments: Lecturers can request adjustments to the timetable if it does not align with their availability or other constraints.

4. Constraint Configuration:

- Custom Constraints: Users can input specific constraints, such as room capacities, lecturer availability, and preferred time slots, which the system considers during timetable generation.
- Dynamic Constraints: The system can handle changes in constraints and update the timetable accordingly.

4.1.3 User Interface (UI) and User Experience (UX)

Login Page : Figure 4.1 shows the smart timetable system login page presents a simple interface for users to access course management, scheduling, and resource allocation. It features a login form with fields for username, password, and role selection, alongside links for password recovery and new account sign-up.

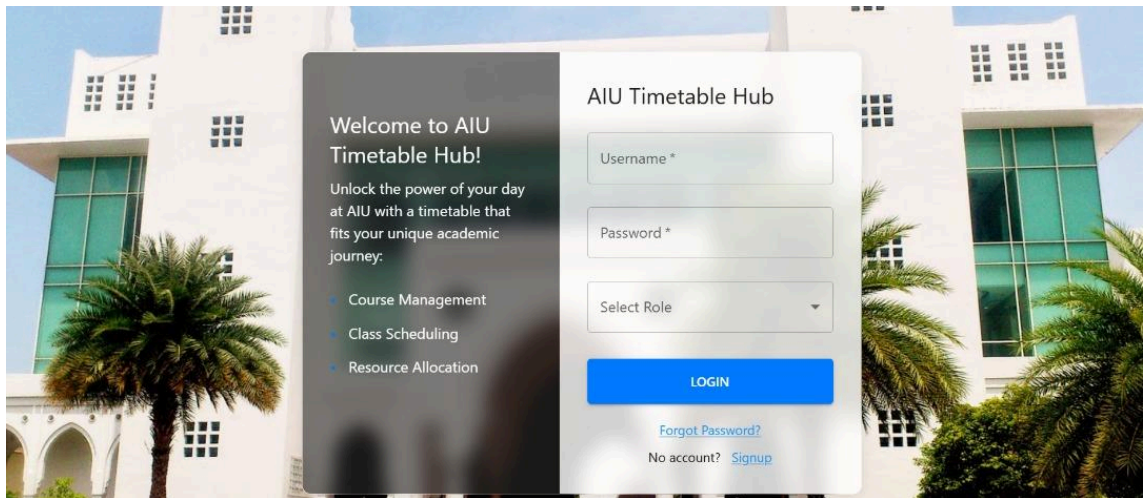


Figure 4.1 Smart Timetable System's Login Page

Admin Dashboard: Figure 4.2 serves as the heart of the system, as it allows the admin to render a lot of activities and take control of the overall system.

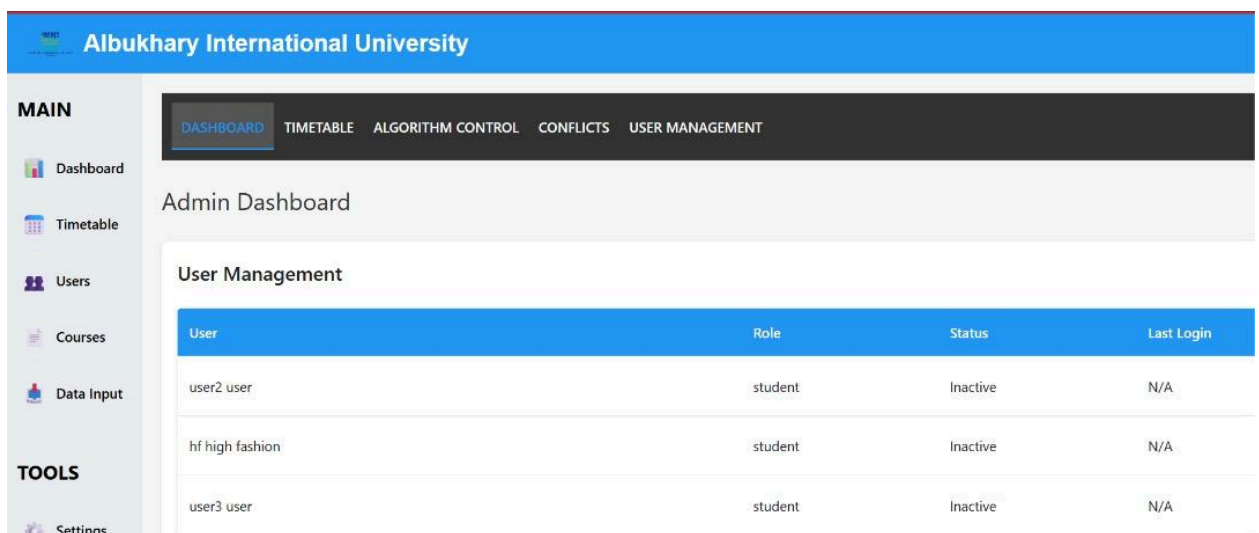


Figure 4.2 Admin Dashboard of smart timetable system

Figure 4.2 has five tabs which includes dashboard, timetable, algorithm control, conflicts, user management. Each tab in the dashboard has its own usage and functionalities.

- **The Dashboard tab(Admin Dashboard):** provides an overview to other tabs. It shows the overview of the functionalities of the admin dashboard.
- **Timetable tab (Admin Dashboard):** Figure 4.3 shows a schedule consisting of class sessions and labs. Admin can view, export, and print the timetable, and filter classes by type, user, and year for customized views. This helps administrators manage and oversee academic schedules efficiently.

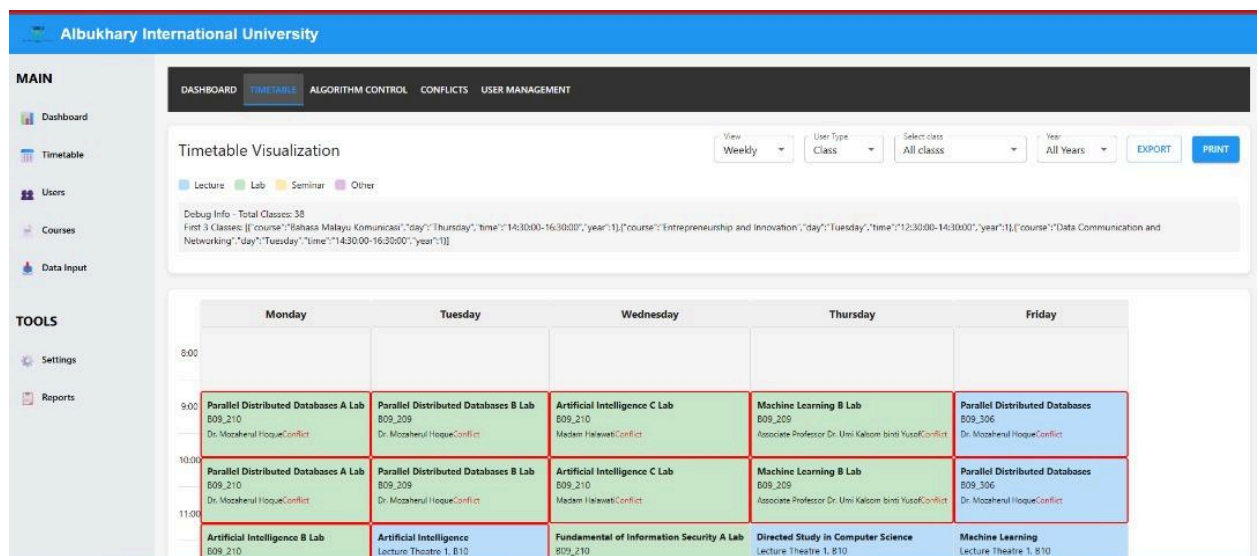


Figure 4.3 Timetable Tab On Admin Dashboard In Smart Timetable System

- **Algorithm Control tab (Admin Dashboard):** Figure 4.4 displays GA Control panel. It allows the admin to adjust parameters like population size, crossover rate, generations, mutation rate, elitism count, and tournament size. There are options to run, stop, and save algorithm parameters, facilitating the optimization of processes using GA.

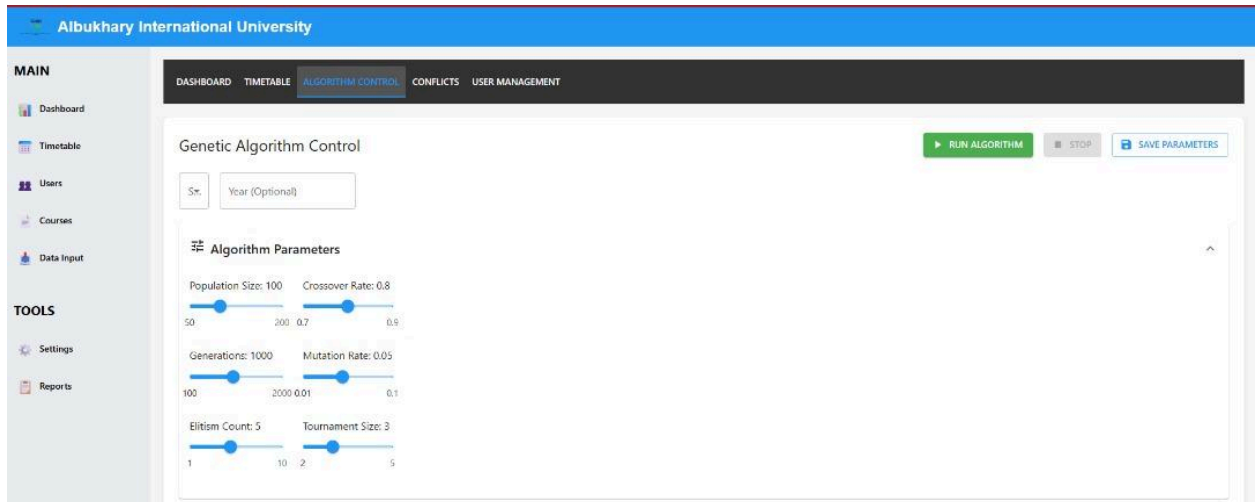


Figure 4.4 Algorithm Control Tab On Admin Dashboard In Smart Timetable System

- **Conflict tab (Admin Dashboard):** Figure 4.5 displays timetable conflicts. Examples of conflicts involve double-booked rooms, overlapping classes.

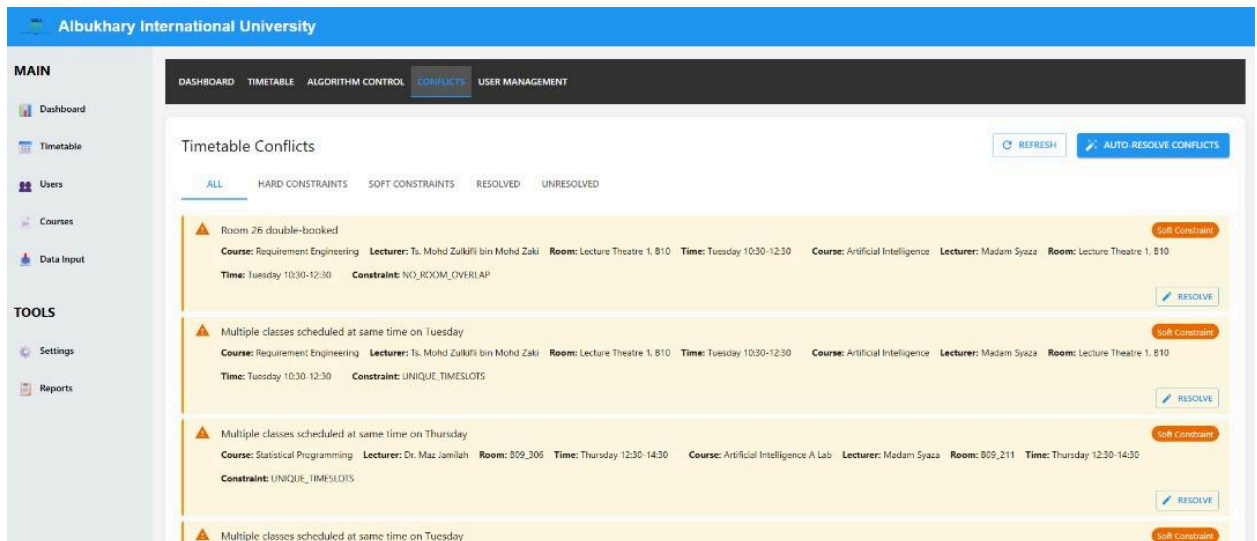
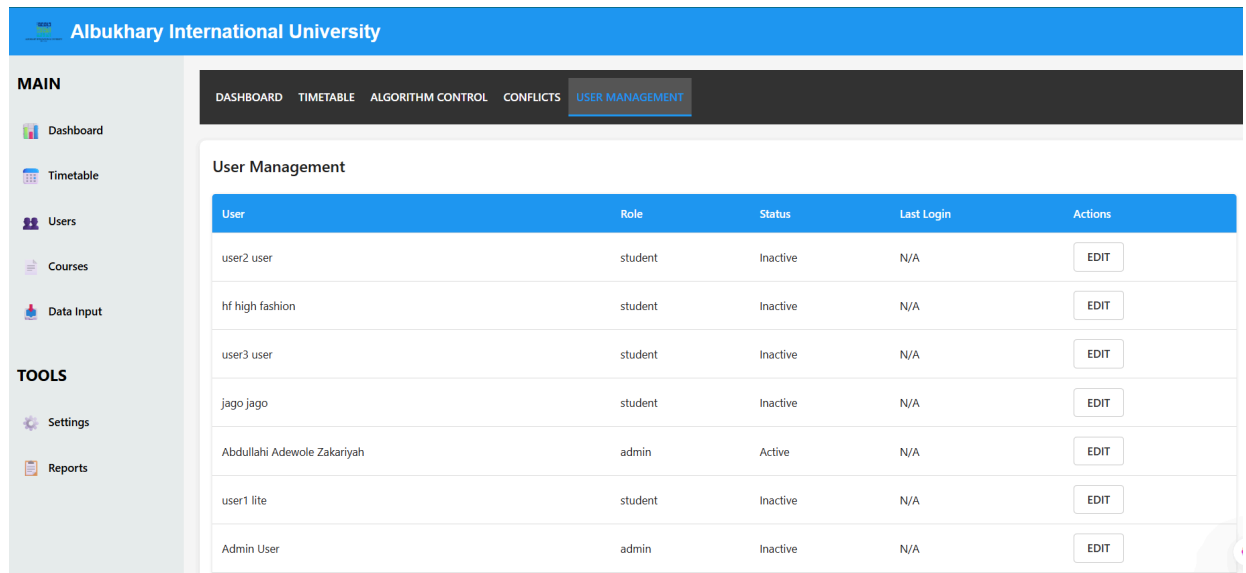


Figure 4.5 Conflict Tab On Admin Dashboard On Smart Timetable System

- **User Management tab** :Figure 4.6 allows the admin to know basic information about the users of the system and also allow the admin to manage the users



User	Role	Status	Last Login	Actions
user2 user	student	Inactive	N/A	EDIT
hf high fashion	student	Inactive	N/A	EDIT
user3 user	student	Inactive	N/A	EDIT
jago jago	student	Inactive	N/A	EDIT
Abdullahi Adewole Zakariyah	admin	Active	N/A	EDIT
user1 lite	student	Inactive	N/A	EDIT
Admin User	admin	Inactive	N/A	EDIT

Figure 4.6 User Management Tabn Smart Timetable System

Student Dashboard :

Figure 4.7 shows a personalized timetable for students. It displays classes and labs scheduled on specific days and times. Students can view their weekly schedule, export it, or print it out, making it easy to manage their academic commitments and plan their week accordingly.

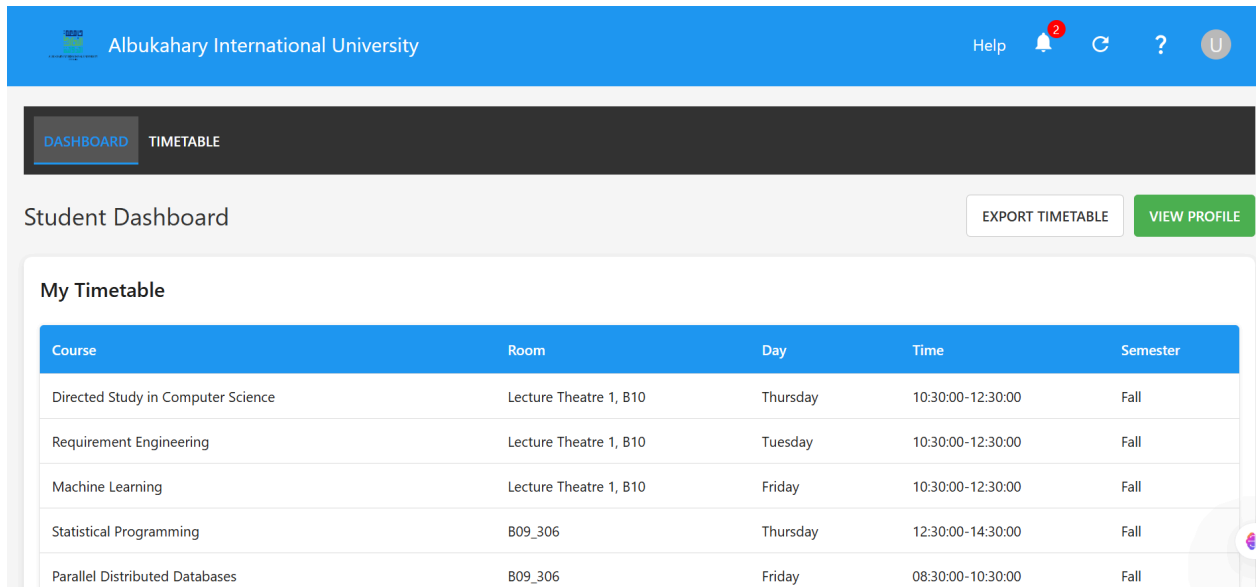


Figure 4.7 Student Dashboard Of Smart Timetable System

- **Timetable tab(Student Dashboard):** Figure 4.8 shows the student personalized schedule, the timetable can export as csv and can be printed making it easy to manage their academic commitments and plan their week accordingly.

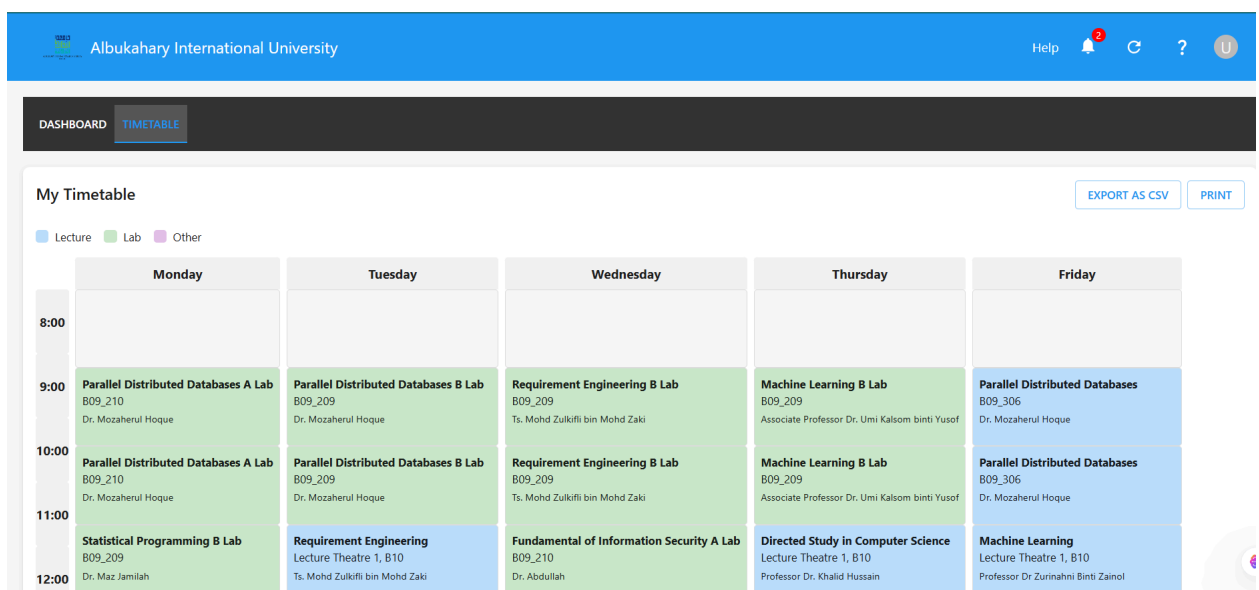
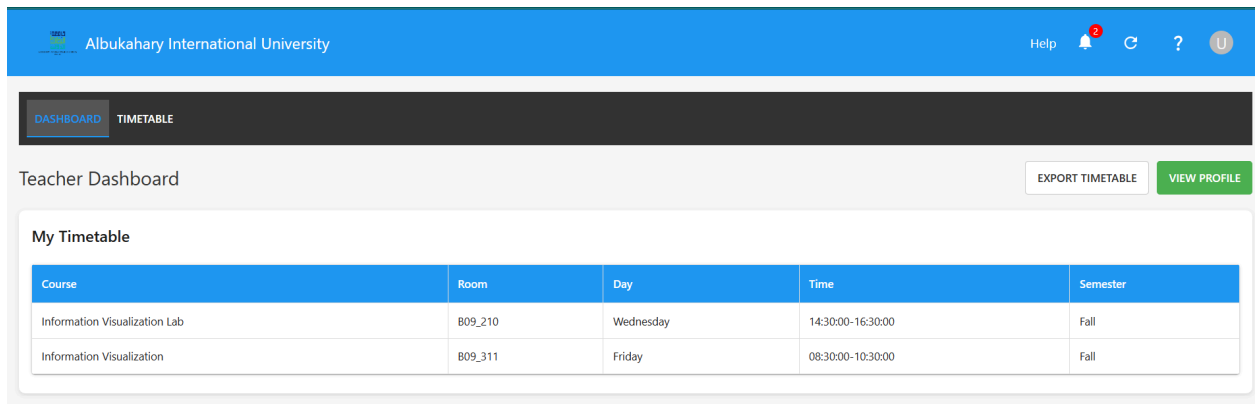


Figure 4.8 Timetable Tab On Student Dashboard

Lecturer Dashboard :Figure 4.9 shows a personalized timetable for the lecturer . It displays classes and labs scheduled on specific days and times. Lecturers can view their weekly schedule, export it, or print it out, making it easy to manage their academic commitments and plan their week accordingly.

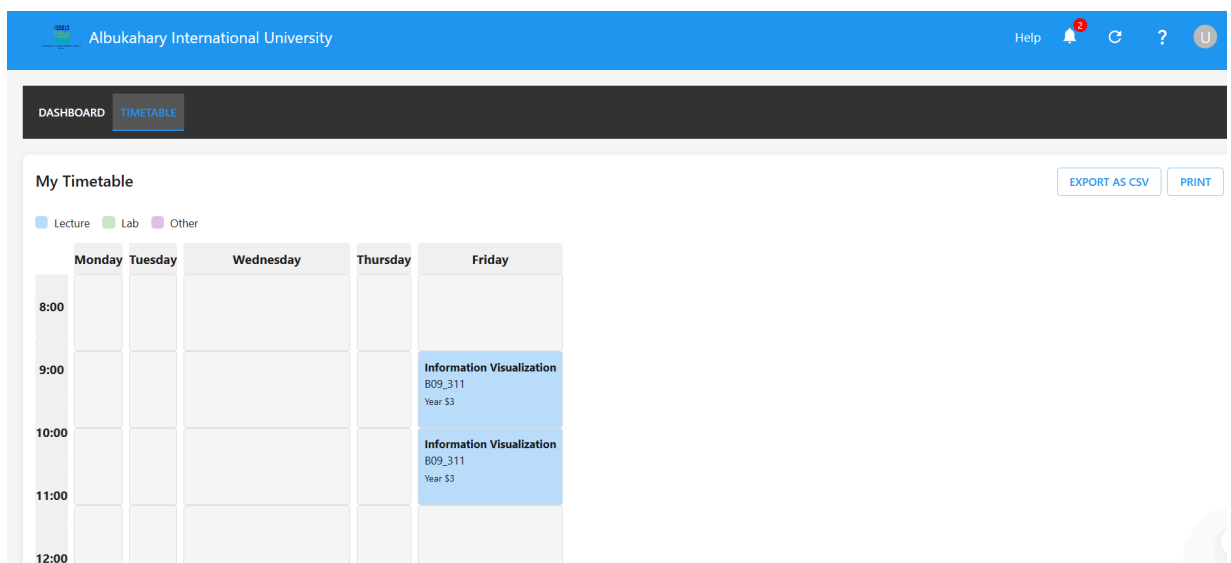


The screenshot shows the 'Teacher Dashboard' with a 'TIMETABLE' tab selected. It features a table titled 'My Timetable' with the following data:

Course	Room	Day	Time	Semester
Information Visualization Lab	B09_210	Wednesday	14:30:00-16:30:00	Fall
Information Visualization	B09_311	Friday	08:30:00-10:30:00	Fall

Figure 4.9 Lecturer dashboard of Smart timetable System

- **Timetable tab (lecturer dashboard)** : Figure 4.10 shows the lecturer personalized schedule, the timetable can export as csv and can be printed making it easy to manage their academic commitments and plan their week accordingly.



The screenshot shows the 'TIMETABLE' tab with a 'My Timetable' section. It includes a legend for Lecture (blue), Lab (green), and Other (purple). The schedule grid shows the following classes:

	Monday	Tuesday	Wednesday	Thursday	Friday
8:00					
9:00					Information Visualization B09_311 Year S3
10:00					Information Visualization B09_311 Year S3
11:00					
12:00					

Figure 4.10 Timetable Tab On Lecturer Dashboard

4.2 Testing

4.2.1 Unit Testing : Unit testing involves testing individual components or modules of the Smart Timetable System in isolation to verify that each performs as designed. The main modules tested were the Admin Module, Lecturer Module, Student Module, and Shared Components (Authentication).

Table 4.1: Unit Test Cases for Smart Timetable System

Test ID	Module	Test case	Input	Expected output	Actual output	Result
UT_001	Admin	Timetable Generation	GA parameters + Valid data	Timetable generated without conflicts	Timetable created successfully	pass
UT_002	Admin	Conflict Auto-resolution	Timetable with conflict	Conflict resolved	Conflict resolved via auto resolve tool	pass
UT_003	Lecturer	View Personalized Timetable	Lecturer login	Correct class schedule displayed	Displayed correctly	pass
UT_004	Student	View and Print Timetable	Student login	Weekly schedule printable/exported	Export and print worked	pass
UT_005	Authentication	New User & Signup and Login	Valid Sign up form	Redirect to login, then dashboard	Success redirect and login	pass

4.2.2 Integration Testing

Integration testing focuses on verifying the interactions between integrated modules of the system. It ensures that data flows correctly between modules and that components such as the backend, frontend, and database communicate accurately.

Table 4.2: Integration Test Cases for Smart Timetable System

Test ID	Modules Involved	Test case	Input/Action	Expected Result	Actual Result	Result
IT_001	Auth+ Dashboard	Role-base login redirection	Login as Admin, Lecturer, Student	Redirect to respective dashboard	Successful redirection	Pass
IT_002	Admin + GA Engine	Timetable Generation with constraints	Set GA config + Run Algorithm	Timetable Generated with zero conflicts	Timetable produced, conflicts detected and fixed	pass
IT_003	Admin + Lecturer	View Assigned classes	Lecturer checks schedule	Classes assigned visible in dashboard	Successfully displayed	pass
IT_004	Admin + conflict resolver	Auto resolve conflicts	Conflict -filled timetable	All detected conflicts autoresolve	All conflicts enforced post-run	pass

4.2.3 System Testing

System testing validates the entire Smart Timetable System end-to-end, from user login to timetable generation and viewing. It simulates real academic scenarios, such as timetable generation for specific semesters, role-based access control, data modification, and schedule export/print functionalities.

Test ID	Scenario	Description	Steps	Expected Result	Actual outcome	Result
ST_001	Full Timetable Generation	Admin generate timetable for year 2 second semester	Login as Admin > configure GA > Run Algorithm	Timetable Generated with valid sessions	Successful generation with 19 classes scheduled	Pass
ST_002	Conflict Detection and Resolution	Detect and resolve hard and soft constraints	View constraints > auto resolve constraints	Missing session scheduled; conflicts respected	All issues resolved, sessions added	pass
ST_003	Student View Synchronization	Check if student see updated schedule after Admin run	Admin runs GA > Student Login > View timetable	Updated schedule shown on the student dashboard	Matched Admin timetable	pass
ST_004	Lecturer View synchronization	Lecturer sees accurate assigned classes	Admin generates > lecturer logs in > Check Dashboard	Lecturer dashboard reflects assigned labs and lectures	Verified correct schedule	pass

Table 4.3: System Test Scenarios and Results

CHAPTER FIVE

CONCLUSION

5.1 Contribution To Social Business/ Social Innovation

Timetable creation is one of the most time-consuming administrative tasks in schools and universities, especially when done manually. The Smart Timetable System was designed to simplify and enhance the accuracy of this process. Instead of relying on manual efforts, the system automatically arranges classes based on available rooms, lecturers, and time slots. This automation helps eliminate scheduling conflicts and allows both students and lecturers to manage their routines more effectively.

By improving class organization, the system contributes to SDG 4: Quality Education, as smoother teaching and learning processes are facilitated. With a clearer schedule, individuals can avoid wasting time on last-minute adjustments.

Additionally, the system supports SDG 9: Industry, Innovation, and Infrastructure, by leveraging advanced technologies such as genetic algorithms and artificial intelligence. These innovations showcase the significant potential of technology in overcoming challenges within education, making systems more intelligent and efficient.

The Smart Timetable System was developed not for commercial purposes, but to address a critical issue in academic planning. With further development, the system could be adapted for use in other departments or smaller educational institutions where timetable management continues to be a significant challenge.

5.2 Future Work

1. **Algorithm Optimization:** Fine-tune the GA parameters (population size, crossover rate, mutation rate) through extensive simulations to identify optimal settings for various scenarios. Explore hybrid approaches combining GA with other optimization techniques to enhance robustness and efficiency.
2. **User Interface Improvements:** Develop a more intuitive and user-friendly interface with features like drag-and-drop scheduling, real-time updates, and interactive visualizations. Allow users to set personalized preferences and constraints for a tailored scheduling experience.
3. **Real-Time Data Handling:** Enhance the system's ability to handle real-time data and dynamic changes, such as last-minute changes in lecturer availability or room bookings. Incorporate machine learning models to predict future scheduling needs based on historical data.
4. **System Integration:** Integrate the Smart Timetable System with other institutional systems for seamless data exchange and consistency. Develop a mobile application version to enable on-the-go access and management of schedules.
5. **Security and Scalability:** Strengthen data security with advanced encryption and multi-factor authentication. Optimize the system architecture for scalability to handle increased loads and larger datasets, ensuring robustness and reliability.
6. **Expansion to Other Sectors:** Explore applications in other sectors like corporate event management, healthcare scheduling, and public transportation. Localize the system to support multiple languages and regional settings for a global audience.
7. **Enhanced Reporting and Analytics:** Develop advanced reporting and analytics features to provide insights into scheduling patterns, resource utilization, and potential areas for improvement. This can help administrators make data-driven decisions to optimize the scheduling process further.
8. **Community and Feedback Integration:** Create a feedback loop where users can provide input on the system's performance and suggest improvements. Engage with the user community to gather requirements and prioritize features, ensuring the system remains aligned with user needs and expectations.

REFERENCES

- Abbaszadeh, M. and S. Saeedvand, 2014. A fast genetic algorithm for solving university scheduling problems. *IAES. Intl. J. Artif. Intell.*, 3: 7-15.
- Alade, O. M., Omidiora, E. O., & Olabiyisi, S. O. (2014). Development of a university lecture timetable using modified genetic algorithms approach. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(9).
https://www.researchgate.net/publication/266674463_C_2014_IJARCSSE_All_Rights Reserved_Development_of_a_University_Lecture_Timetable_using_Modified_Genetic_Algorithms_Approach
- Alba, E., Chicano, F., & Luque, G. (Eds.). (2017). *Smart cities: Second international conference, Smart-CT 2017, Málaga, Spain, June 14-16, 2017, proceedings* (1st ed.). Springer.
- Arogundade, O. T., Akinwale, A. T., & Aweda, O. M. (2010). A genetic algorithm approach for a real-world university examination timetabling problem. *International Journal of Computer Applications*, 12(5), 0975-8887.
- Babaagba, K. O., & Arekete, S. A. (2017). A review of agent-based university course timetabling systems. *International Journal of Engineering Development and Research*, 5(2), 566–568. © 2017 IJEDR | ISSN: 2321-9939. Retrieved from <https://rjwave.org/ijedr/papers/IJEDR1702097.pdf>
- Barhate, H. B., Wasnik, D. A., Bhombe, S. N., & Sapkale, S. S. (2024). Smart timetable generator. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 4(3), 554–555. <https://doi.org/10.48175/IJARSCT-17285>
- Beligiannis, G. N., Moschopoulos, C., & Likothanassis, S. D. (2009). A genetic algorithm approach to school timetabling. *Journal of the Operational Research Society*, 60(1), 23-42.
- Brown, A., & Kumar, R. (2021). Challenges in the implementation of AI-driven scheduling systems. *Journal of Educational Technology*, 12(3), 45-59.
- Central Statistics Office. (2024). *Smart technology 2024: Key findings*. Online ISSN: 2990-8833. Retrieved from

<https://www.cso.ie/en/releasesandpublications/ep/p-smrt/smarttechnology2024/keyfindings/>

Colomi, A., Dorigo, M., & Maniezzo, V. (1992). A genetic algorithm to solve the timetable problem. Politecnico di Milano, Milan, Italy TR, 90-060.

Herath, A. K. (2017). Genetic Algorithm For University Course Timetabling Problem. *eGrove*.

<https://egrove.olemiss.edu/cgi/viewcontent.cgi?article=1442&context=etd>

Cotta, C., & van Hentenryck, P. (2006). Genetic algorithms for the university course timetabling problem. *European Journal of Operational Research*, 171(3), 670–684.

<https://doi.org/10.1016/j.ejor.2005.03.027>

Chai, S. Y. L., & Tyng, L. Y. (2023). Auto timetable management mobile application. *Trends in Undergraduate Research*, 6(2), c1–c12. <https://doi.org/10.33736/tur.5730.2023>.

Chavan, P. B., Patil, S., Garadi, R., Jadhav, R., & Ghotane, S. (2024). Smart digital timetable. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 4(5).

Chuenkrut, W., & Achayuthakan, P. (2019). The efficiency of using a class timetable management database system, Faculty of Science and Technology, Suan Sunandha Rajabhat University. *International Academic Multidisciplinary Research Conference in Amsterdam 2019*, 1–10.

Firke, R., Bhabad, P., Gangarde, O., Magar, A., & Tawlare, A. (2023). Automatic timetable generation system. *IJCRT: International Journal of Creative Research Thoughts*, 11(10), 1–10. <https://www.ijcrt.org>

Gen, M., & Lin, L. (2023). *Genetic algorithms and their applications*. In M. Gen & L. Lin (Eds.), *Springer handbook of engineering statistics* (pp. 635–674).

Hassani, M. S. A., & Habibi, F. (2013). Solution approaches to the course timetabling problem. *Artificial Intelligence Review*, 39(2), 133–149.

Herath, A. K. (2017). *Genetic algorithm for university course timetabling problem* (Master's thesis). University of Mississippi.

<https://egrove.olemiss.edu/cgi/viewcontent.cgi?article=1442&context=etd>

Jones, M., & Taylor, P. (2019). The rise of smart timetable systems in educational institutions. *Technology in Education Review*, 28(2), 102-118.

Kavade, R., Qureshi, S., Veer, N., Ugale, V., & Agrawal, P. (2023). Smart timetable system using AI and ML. *International Journal of Creative Research Thoughts (IJCRT)*, 11(5), 1–5. Retrieved from <http://www.ijcrt.org>

Kumar, M., Husain, D. M., Upreti, N., & Gupta, D. (2010). Genetic algorithm: Review and application. Available at SSRN 3529843.

Kumar, S., Gupta, R., & Sharma, M. (2022). Machine learning in education: Applications and challenges in timetable generation. *International Journal of Computer Science*, 45(1), 67–89. Retrieved from <https://example.com/ml-in-education>

Lee, K. (2022). Enhancing scheduling efficiency with automated systems. *Educational Innovations Journal*, 19(4), 76-89.

Lee, C., Kim, J., & Park, S. (2019). Optimizing academic schedules using artificial intelligence: A case study. *Journal of Educational Technology*, 34(2), 112–124. Retrieved from <https://example.com/academic-schedules-ai-case-study>

Liew, S. P. (2005). *General timetabling system for school* (Bachelor's thesis). Universiti Malaysia Sarawak, Faculty of Computer Science and Information Technology.

Limkar, S., Khalwadekar, A., Tekale, A., Mantri, M., & Chaudhari, Y. (2015). Genetic algorithm: Paradigm shift over a traditional approach of timetable scheduling. In S. Satapathy, B. Biswal, S. Udgata, & J. Mandal (Eds.), *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications* (pp. 771–780). Springer. https://doi.org/10.1007/978-3-319-11933-5_87

- Mohammed, M. A., Khanapi, M., Ghani, A., Obaid, O. I., & Mostafa, S. (2017). A review of genetic algorithm application in examination timetabling problem. *Journal of Engineering and Applied Sciences*, 12(20), 5166-5181.
- Muñoz, A., García, P., & Torres, R. (2020). Challenges in manual scheduling for educational institutions: A review of current practices. *International Journal of Educational Management*, 18(3), 203–215. Retrieved from <https://example.com/challenges-manual-scheduling>
- M, N. S., & Vadivel, N. R. (2024). Smart timetable display for students' subject allocation in classroom using IOT. *World Journal of Advanced Research and Reviews*, 21(3), 1269–1275. <https://doi.org/10.30574/wjarr.2024.21.3.0810>
- Obaid, O. I., Mohammed, M. A., & Ahmad, M. S. (2015). Solving examination timetabling problem by using genetic algorithm. OmniScriptum Publishing.
- Oladimeji, I. W. (2020). *Automated lectures-based timetabling generation using evolutionary algorithm*. *LAUTECH Journal of Computing and Informatics*, 1(1), 1–x. <https://www.laujci.lautech.edu.ng>
- Ogunseye, J. O., & Ojuawo, O. O. (2022). Automated lecture timetable generation using genetic algorithm. *FEPI-JOPAS*, 4(2), 79–84.
- Park, J., & Kim, K. Y. (2017). Meta-modeling using generalized regression neural network and particle swarm optimization. *Applied Soft Computing*, 51, 354–369.
- Patel, R., & Singh, P. (2018). Genetic algorithms in scheduling problems: A review. *International Journal of Computational Intelligence*, 12(4), 213–224. <https://doi.org/10.1007/s10462-018-9752-4>
- Roberts, L., & Harris, G. (2020). Barriers to adopting smart scheduling technologies in universities. *Journal of Educational Administration*, 14(1), 67-82.

Ross, R. L., & Vázquez, A. (2020). Application of genetic algorithms in educational scheduling: A case study. *Procedia Computer Science*, 174, 374–383.

<https://doi.org/10.1016/j.procs.2020.06.048>

School Timetable - QuickSchools. (n.d.). QuickSchools.

<https://www.quickschools.com/quickschools/features/scheduling>

Shorman, S. M., & Pitchay, S. A. (2015). Significance of parameters in genetic algorithm, the strengths, its limitations and challenges in image recovery. *ARP Journal of Engineering and Applied Sciences*, 10(2), 585-590.

https://d1wqtxts1xzle7.cloudfront.net/45994039/SIGNIFICANCE_OF_PARAMETERS_IN_GENETIC_ALGORITHM_THE_STRENGTHS_ITS_LIMITATIONS_AND_CHALLENGES_IN_IMAGE_RECOVERY-libre.pdf

Singh, P., & Patel, R. (2020). Automation in academic scheduling: Benefits and challenges. *International Journal of Educational Research*, 31(5), 56-72.

Smith, J. (2021). AI-based scheduling: The future of academic timetabling. *Journal of AI and Education*, 8(1), 34-50.

Smith, A., & Taylor, R. (2021). AI-driven solutions for resource optimization in schools: An emerging paradigm. *Advances in Computer Applications*, 27(5), 345–360. Retrieved from <https://example.com/ai-resource-optimization-schools>

Sneha, M., & Vadivel, R. (2024). Smart timetable display for students' subject allocation in classroom using IoT. *World Journal of Advanced Research and Reviews*, 21(3), 1269–1275. <https://doi.org/10.30574/wjarr.2024.21.3.0810>

Taylor, J., & Green, B. (2022). Predictive analytics in education: The next step in smart timetables. *Journal of Learning Technologies*, 15(2), 23-39.

Thakare, S., Nikam, T., & Patil, M. (2020). Automated timetable generation using genetic algorithm. *International Journal of Engineering Research & Technology (IJERT)*, 9(07).

<http://www.ijert.org>

Wang, J., & Liu, Y. (2021). A hybrid genetic algorithm for the university course timetable problem. *Journal of Computational Science*, 50, 101200.
<https://doi.org/10.1016/j.jocs.2020.101200>

Williams, S. (2021). Flexibility in scheduling: How smart systems adapt to change. *Higher Education Technology Journal*, 24(3), 89-101

Yılmaz, F., & Tetik, S. (2019). Timetable optimization in educational institutions using genetic algorithms. *International Journal of Educational Management*, 33(5), 890–907.
<https://doi.org/10.1108/IJEM-02-2018-0035>

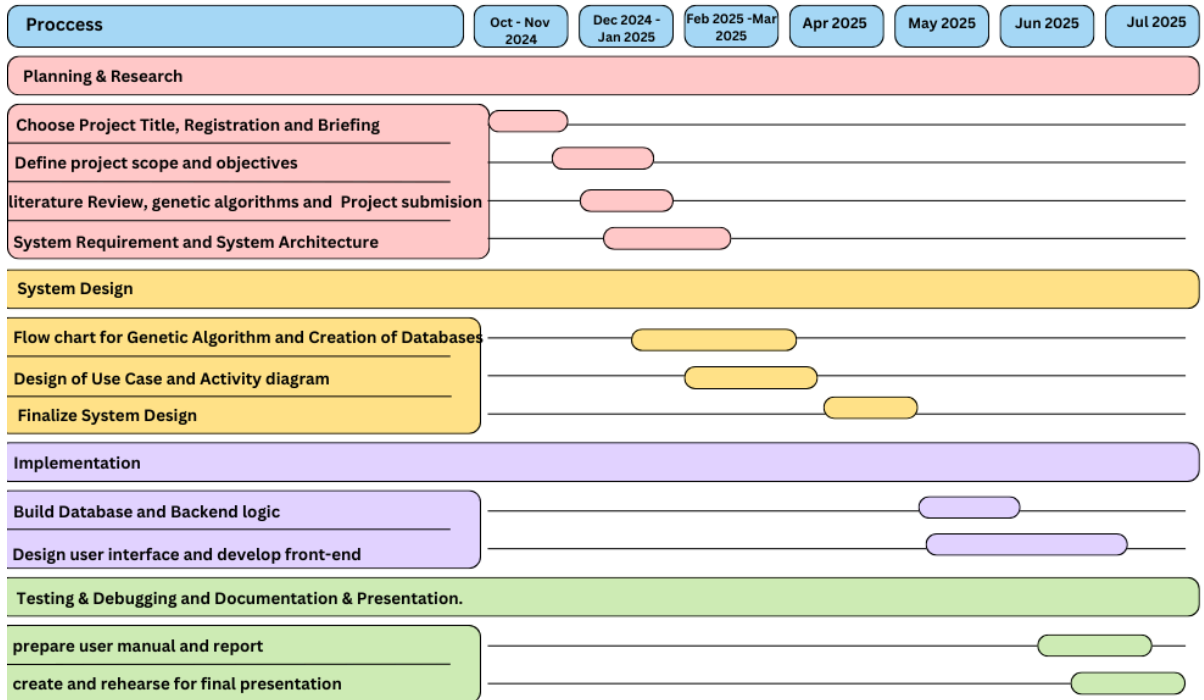
Yua, B., Yanga, Z., Suna, X., Yao, B., Zenga, Q., & Jeppesen, E. (2011). Parallel genetic algorithm in bus route headway optimization. *Transportation Research Part C: Emerging Technologies*, 19(6), 1032–1046. <https://doi.org/10.1016/j.trc.2011.03.002>

Zainuddin, H. N. (2006). Genetic Algorithm Application in Timetabling (Doctoral dissertation, Jabatan Kejuruteraan Elektrik, Fakulti Kejuruteraan, Universiti Malaya).

APPENDIX A

1. Gantt Chart

SMART TIMETABLE SYSTEM



APPENDIX B

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.



Page 2 of 45 - Integrity Overview

Submission ID trn:oid::9832:79932298





8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Quoted Text
- Crossref database
- Crossref posted content database

Match Groups

-  **52 Not Cited or Quoted 8%**
Matches with neither in-text citation nor quotation marks
-  **5 Missing Quotations 1%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 4%  Internet sources
- 0%  Publications
- 7%  Submitted works (Student Papers)

APPENDIX C

```
@dataclass
class TimeSlot:
    day: str
    start_time: time
    end_time: time
```

```
@dataclass
class ScheduleItem:
    course_id: str
    course_name: str
    lecturer_id: str
    lecturer_name: str
    room_id: str
```

```
room_name: str
day: str
start_time: time
end_time: time
timeslot_id: Optional[str] = None
```

```
@dataclass
```

```
class Conflict:
```

```
    type: str
    description: str
    items: List[ScheduleItem]
    constraint: Optional[str] = None
```

```
DAYS = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
```

```
STARTING_HOUR = 8
```

```
STARTING_MINUTE = 30
```

```
ENDING_HOUR = 18
```

```
ENDING_MINUTE = 30
```

```
PERIOD_DURATION = 120
```

```
AVAILABLE_MINUTES = (ENDING_HOUR * 60 + ENDING_MINUTE) -  
(STARTING_HOUR * 60 + STARTING_MINUTE)
```

```
PERIODS_PER_DAY = AVAILABLE_MINUTES // PERIOD_DURATION
```

```
MAX_GENERATIONS_WITHOUT_IMPROVEMENT = 50
```

```
HARD_CONSTRAINT_PENALTY = 1000.0
```

```
SOFT_CONSTRAINT_PENALTY = 1.0
```

```
class Chromosome:
```

```
    def __init__(self, schedule_items: List[ScheduleItem] = None):  
        self.schedule_items = schedule_items or []  
        self.fitness = 0.0  
        self.hard_violations = 0  
        self.soft_violations = 0
```



```

self.conflicts = []

def copy(self):
    new_chromosome = Chromosome(copy.deepcopy(self.schedule_items))
    new_chromosome.fitness = self.fitness
    new_chromosome.hard_violations = self.hard_violations
    new_chromosome.soft_violations = self.soft_violations
    new_chromosome.conflicts = copy.deepcopy(self.conflicts)
    return new_chromosome

class TimetableGenerator:
    def __init__(self, db: Session, semester: str, year=None,
                 population_size=50, max_generations=100, crossover_rate=0.8,
                 mutation_rate=0.05, elitism_count=5, tournament_size=5):
        self.db = db
        self.semester = semester
        self.year = year
        self.population_size = population_size
        self.max_generations = max_generations
        self.crossover_rate = crossover_rate
        self.mutation_rate = mutation_rate
        self.elitism_count = elitism_count
        self.tournament_size = tournament_size
        self.lecturers = self._load_lecturers()
        self.courses = self._load_courses(year=year)
        self.rooms = self._load_rooms()
        self.constraints = self._load_constraints()
        self.course_lecturer_mapping = self._create_course_lecturer_mapping()
        self.population = []
        self.hard_constraints = [c for c in self.constraints if c.constraint_id.startswith('HC')]
        self.soft_constraints = [c for c in self.constraints if c.constraint_id.startswith('SC')]

    def _load_lecturers(self) -> Dict[str, Lecturer]:

```

```

lecturers = {}
db_lecturers = self.db.query(Lecturer).all()
for lecturer in db_lecturers:
    lecturers[lecturer.lecturer_id] = lecturer
    if not hasattr(lecturer, 'courses'):
        lecturer.courses = []
    if hasattr(lecturer, 'course_id') and lecturer.course_id:
        lecturer.courses.append(lecturer.course_id)
return lecturers

def _load_courses(self, year=None):
    courses = {}
    query = self.db.query(Course).filter(Course.semester == self.semester)
    if year is not None and year > 0:
        query = query.filter(Course.year == year)
    db_courses = query.all()
    for course in db_courses:
        courses[course.course_id] = course
        sessions_count = max(1, course.credit // 2) if hasattr(course, 'credit') else 1
        course.sessions_count = sessions_count
    return courses

def _load_rooms(self) -> Dict[str, Room]:
    rooms = self.db.query(Room).all()
    return {room.room_id: room for room in rooms}

def _load_constraints(self) -> List[Constraint]:
    constraints = self.db.query(Constraint).all()
    return constraints

def _create_course_lecturer_mapping(self) -> Dict[str, str]:
    mapping = {}

```

```

for course_id, course in self.courses.items():
    if hasattr(course, 'lecturer_id') and course.lecturer_id:
        mapping[course_id] = course.lecturer_id
for lecturer_id, lecturer in self.lecturers.items():
    if hasattr(lecturer, 'course_id') and lecturer.course_id:
        course_ids = lecturer.course_id if isinstance(lecturer.course_id, list) else
[lecturer.course_id]
        for course_id in course_ids:
            mapping[course_id] = lecturer_id
    if hasattr(lecturer, 'courses') and lecturer.courses:
        for course_id in lecturer.courses:
            mapping[course_id] = lecturer_id
return mapping

def initialize_population(self):
    self.population = [self.create_random_chromosome() for _ in range(self.population_size)]

def create_random_chromosome(self) -> Chromosome:
    chromosome = Chromosome()
    lecturer_schedule = {l_id: {day: [] for day in DAYS} for l_id in self.lecturers}
    room_schedule = {r_id: {day: [] for day in DAYS} for r_id in self.rooms}
    student_schedule = {}
    course_order = sorted(self.courses.keys(), key=lambda c_id:
self.courses[c_id].no_of_students, reverse=True)
    lab_rooms = ["B09_209", "B09_210", "B09_211"]

    for course_id in course_order:
        course = self.courses[course_id]
        sessions_needed = getattr(course, 'sessions_count', 1)
        student_group = course.student_group if hasattr(course, 'student_group') else course_id
        if student_group not in student_schedule:
            student_schedule[student_group] = {day: [] for day in DAYS}

```

```

assigned_lecturer_id = self.course_lecturer_mapping.get(course_id)
is_lab_course = "Lab" in course.course_name
is_lettered_course = any(suffix in course.course_name.split() for suffix in ["A", "B", "C",
"D"])

suitable_rooms = lab_rooms if is_lab_course else [r_id for r_id, r in self.rooms.items() if
r.capacity >= course.no_of_students]
if is_lettered_course and random.random() < 0.7:
    suitable_rooms = lab_rooms

for _ in range(sessions_needed):
    for _ in range(200): # Max attempts
        day = random.choice(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"])
        period = random.randint(1, PERIODS_PER_DAY)
        start_time, end_time = period_to_time(period)
        new_timeslot = TimeSlot(day=day, start_time=start_time, end_time=end_time)

        if not any(timeslots_overlap(new_timeslot, slot) for slot in
lecturer_schedule[assigned_lecturer_id][day]):
            room_id = next((r_id for r_id in suitable_rooms if not
any(timeslots_overlap(new_timeslot, slot) for slot in room_schedule[r_id][day])), None)
            if room_id and not any(timeslots_overlap(new_timeslot, slot) for slot in
student_schedule[student_group][day]):
                chromosome.schedule_items.append(ScheduleItem(
                    course_id=course_id, course_name=course.course_name,
                    lecturer_id=assigned_lecturer_id,
lecturer_name=self.lecturers[assigned_lecturer_id].lecturer_name,
                    room_id=room_id, room_name=self.rooms[room_id].room_name, day=day,
start_time=start_time, end_time=end_time
                ))
            lecturer_schedule[assigned_lecturer_id][day].append(new_timeslot)

```

```
room_schedule[room_id][day].append(new_timeslot)
student_schedule[student_group][day].append(new_timeslot)
break
return chromosome
```