applied sciences

MDPI

Article

Optimizing the Scheduling of Teaching Activities in a Faculty

Francis Patrick Diallo [1,2] and Cătălin Tudose [1,2,*]

[1] Faculty of Automatic Control and Computers, National University of Science and Technology POLITEHNICA Bucharest, 060042 Bucharest, Romania; francis.diallo@stud.acs.upb.ro
[2] Luxoft Romania, 020335 Bucharest, Romania
* Correspondence: catalin.tudose@gmail.com

Abstract: To maximize resource usage, minimize disputes, and improve academic experience, professors must schedule teaching activities efficiently. This study provides an optimized automated schedule creation technique. The system generates schedules that aim to be conflict-free and efficient, utilizing evolutionary algorithms along with multi-objective optimization. Resource usage, scheduling problems, and faculty/student satisfaction are the goals of the research. The system optimizes scheduling based on room limitations, instructor availability, and student preferences. The project uses system design, model creation, algorithm implementation, and performance analysis to solve the difficult timetable-generating problem. This research should save administrators time, improve academic operations, and improve staff and student academic experiences. Scalability and flexibility allow the system to be used in multiple faculties and incorporate new limits and requirements. This paper presents a complete approach to faculty scheduling, including insights and recommendations for future study and application in educational institutions.

Keywords: automatic timetable generation; faculty scheduling; metaheuristics algorithms; resource utilization; scheduling conflicts; scalability; Timefold

Citation: Diallo, F.P.; Tudose, C. Optimizing the Scheduling of Teaching Activities in a Faculty. *Appl. Sci.* **2024**, *14*, 9554. https://doi.org/10.3390/app14209554

Academic Editor: Luis Javier Garcia Villalba

Received: 6 September 2024
Revised: 15 October 2024
Accepted: 17 October 2024
Published: 19 October 2024

1. Introduction

During an academic year, universities invest significant time and resources into manually creating schedules that comply with regulations and meet the needs of all stakeholders. Scheduling courses and exams in higher education is a complex challenge for institutions worldwide, requiring the optimal use of limited resources. In today's climate, reducing operational expenses is essential, making manual timetable development seem inefficient when it can be automated and optimized.

This research aims to design and implement an automated system that generates schedules for academic activities within a faculty. The process involves efficiently assigning courses, faculty, and rooms to available time slots while avoiding conflicts. By automating scheduling, the faculty can optimize resource usage, reduce scheduling conflicts, and streamline its processes.

1.1. Context

Efficient scheduling is essential to maintaining an organized academic environment, particularly in faculties with numerous courses, faculty members, and students. The manual process is time-consuming, error-prone, and often leads to suboptimal outcomes. This project tackles these issues by applying advanced algorithms and technology to automate timetable generation, drawing inspiration from the well-known Timetable Problem [1].

1.2. Motivation

Organizing educational activities is a complex problem with broader applications across various sectors such as healthcare, transportation, and entertainment, where resource allocation and scheduling limitations present significant challenges [2,3]. These issues fall

into the NP-complete category, meaning optimal solutions are difficult to find efficiently [4], and heuristic approaches are often employed to generate practical results. This research aims to improve scheduling at universities by automating the process, addressing the limitations of manual methods, and optimizing resource allocation, conflict minimization, and satisfaction among stakeholders.

Traditional approaches demand considerable time and effort from administrators and faculty, often resulting in schedules that fail to optimally allocate resources, avoid conflicts, or meet the preferences of all involved. By automating the process, the system will create timetables that better allocate resources, reduce scheduling conflicts, and improve satisfaction for both students and faculty.

1.3. Objectives

The main objective is to create and execute a resilient system that automates the process of generating timetables for the faculty. This will be achieved through the development of an Automated Timetable Generation System. The system will consider multiple constraints, preferences, and optimization criteria to generate timetables of superior quality.

The system should have the capability to optimize the use of time slots, faculty, and rooms, ensuring better resource management and minimizing potential conflicts. By automating course allocation, the project aims to reduce instances of scheduling clashes, such as faculty members being assigned to multiple classes simultaneously or students having overlapping courses.

Additionally, the system will also enhance the satisfaction of both students and faculty by taking into account their preferences and availability, ultimately leading to a more efficient and well-coordinated academic environment.

1.4. Expected Impact

The anticipated benefits of implementing an automated timetable generation system within the faculty are manifold and significant. Automation will reduce the time required by administrative staff and faculty, allowing them to focus on more strategic activities. Optimizing timetables can increase efficiency in resource utilization, reduce scheduling conflicts, and streamline administrative processes. The system will efficiently allocate courses, faculty members, and rooms, mitigating inefficiencies and minimizing underutilization or overbooking.

The implementation of an automated timetable generation system will also enhance the academic experience by promoting a better work–life balance for faculty members. This will be achieved by aligning their schedules with their preferences and workload constraints, preventing scheduling conflicts. The system is designed to adapt to growing faculties and new courses, accommodating evolving academic requirements while ensuring scalability.

Additionally, the system will manage various scheduling formats, including semester-based or modular systems, and handle a range of academic activities such as lectures, seminars, laboratory sessions, and exams (with a focus on weekly scheduling). This flexibility ensures the system's applicability across different faculties within the institution.

2. Related Studies

Over time, educational institutions around the globe have worked extensively on the deployment of ever more-effective algorithms to alleviate the administrative portion that occurs annually with technological advancement, and numerous scientific articles have been created based on these technical achievements.

For instance, D'souza et al. [5] used the Genetic Algorithm meta-heuristic algorithm, which is based on Darwin's theory of evolution, and parameters, including Course code, number of hours for each course, teacher code, class code, room code, day, and time, to generate a timetable in matrix form. In this study, the authors did not emphasize limitations or the implementation of a functioning platform, but they did illustrate the computing performance of this approach.

Research conducted at the University of Technology Malaysia [6] highlighted the constraints that can reduce the performance of the optimization algorithm to generate a stable schedule and proposed an algorithm that is based on the pre-processing of the matrix to be processed and the use of the Neighborhood Search Algorithm, which is based on a mathematical function.

Wungguli and Nurwan [7] created a course scheduling optimization model using Integer Linear Programming [8] to reduce lecturers' and students' dissatisfaction. This purpose is achieved by the model's objective function, which seeks to minimize the amount of student and lecturer rejection of the schedule based on the formulation of the equation's time option.

Feng, Lee, and Moon [9] developed an algorithm based on a combination of their own mathematical model and the Hybrid Genetic Algorithm after comparing the performance and parameters of other optimization algorithms.

Schaerf [10] provided a comprehensive overview of the different types of timetable optimization problems, as well as a summary of the most frequent methods used to solve them. One of the methods is direct heuristics, which involves filling the full schedule with a single course (or one set of courses) at a given moment if there are no scheduling conflicts. The approaches employed in this problem include reduction to graph coloring, network flow problem, and simulated annealing. "Simulated annealing is a probabilistic method" [11] used to identify the global minimum or maximum of a cost function, which might include several local minimums or maximums. "Tabu search is a local search method" [12] that is widely recognized as a powerful tool for tackling challenging optimization issues. On the other hand, the rule-based approach consists of five sorts of rules: allocation rules, restriction rules, local change rules, and contextual rules.

Socha, Sampels, and Manfrin [13] studied several variants of the Ant Colony Optimization technique that were applied to a set of predefined restrictions.

An additional approach provided by Abuhamdah and Ayob [14] is the use of a computational scheduling system. MPCA-ARDA is the hybridization of the Multi-Neighborhood Particle Collision Algorithm [15] with the Adaptive Randomized Descent Algorithm acceptance criteria [16]. It was developed to address problems related to organizing courses at universities. This method was studied and compared to the algorithms in their simplest form, and it was shown to be more effective.

Shrunkhala et al. [17] used Genetic Algorithm to create a platform that involves a dashboard that students can use to navigate the educational institute's platform, job allocation. Ong [18] created a platform using PHP, Laravel, and SQL to generate a schedule for students and professors automatically.

Yekta and Day [19] focused mainly on applying a combination of algorithms, such as matching algorithms, second-price concepts, and improvement, to promote truth-telling among agents with limited logic. They compared these algorithms based on metrics of equity, efficacy, and incentive integration to evaluate their effectiveness.

Burgess [20] examined the utilization of OptaPlanner [21], an open-source constraint solver and predecessor to Timefold Solver [22], to create an automated course scheduling system for the Computer Science Department at California State University. The study focuses on the complexity of manual scheduling by utilizing advanced algorithms to optimize course assignments, ensure balanced faculty workloads, and meet student requirements. The results indicate significant improvements in the effectiveness and efficacy of scheduling, highlighting the potential of OptaPlanner in handling difficult scheduling problems in academic settings with a smaller dataset (70 courses, 50 instructors, 20 rooms, and 60 timeslots).

Fiechter [23] tackles the complex problem of generating university timetables that comply with various constraints, such as room boundaries, teacher availability, and course prerequisites. By applying the algorithms of the Optaplaner API, Fiechter successfully generated timetables that not only satisfied all hard restrictions but also optimized multiple soft constraints, resulting in increased satisfaction among both teachers and students. The

issue instance examined by Fiechter consisted of a small dataset covering three semesters. It involved scheduling 14 courses and 70 lectures within a school week that consisted of 40 periods (with 8 time slots each day) and had 8 accessible rooms. The resulting schedules were generated within approximately 7 s and were found to be comparable to the existing timetables currently being utilized for the semester. This exemplifies the efficacy and pragmatic utility of Fiechter's technique in real-life contexts.

However, none of the aforementioned studies have explored or addressed the complexities involved in scenarios with a significantly larger scale, such as those involving thousands of courses, teachers, student groups, rooms, and time slots. The existing research primarily focuses on smaller datasets with relatively limited numbers of variables, which might not capture the full spectrum of challenges faced by large educational institutions. In contrast, the present study undertakes the challenge of optimizing timetables for extensive datasets, addressing a broader range of constraints, and ensuring the scalability of the solution. This effort marks a significant advancement in the application of algorithmic approaches to timetable generation, catering to the needs of institutions with substantial student and faculty populations.

3. Model of the Automatic Timetable Generation System

3.1. System Architecture

A web-based platform is proposed for the development of an Automatic Timetable Generation system for a faculty. The frontend of the system is constructed using the Angular framework, while the backend is developed using Java with Spring Boot. The aforementioned architectural approach guarantees a design that is both scalable and modular, effectively segregating the presentation layer from the business logic and data processing layers, as shown in Figure 1. The proposed system is designed to adopt a client–server architecture, wherein the Angular-based frontend can establish communication with the Java-based backend via APIs.

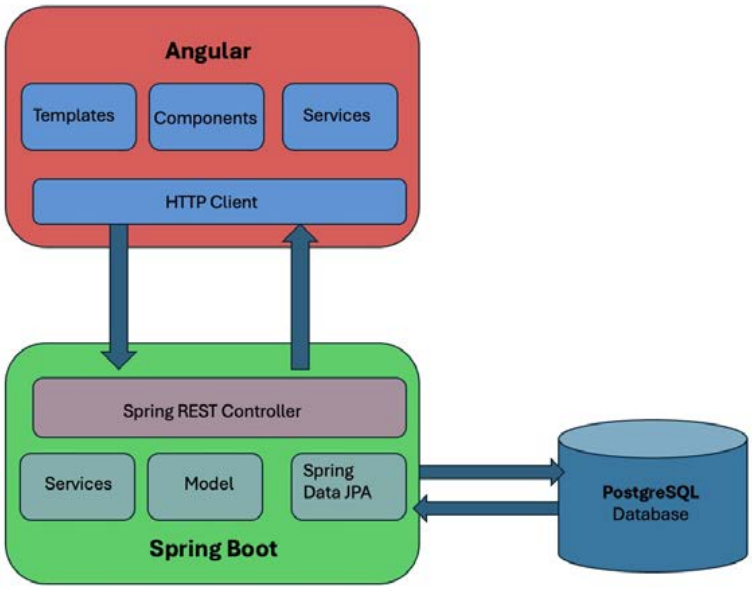

Figure 1. Application architecture.

The frontend of the system uses Angular [24], a framework for building single-page applications. It employs a component-based architecture [25], allowing for reusable and modular code. Key Angular features include data binding [26], dependency injection [26], routing [25], reactive programming [25], and robust testing support [26]. The frontend manages modules such as user authentication and timetable display, interacting with backend application programming interfaces (APIs) to retrieve data and update the application's status, guaranteeing a user interface that is dynamic and responsive.

The backend is built with Java [27] and the Spring Boot framework [28]. It features auto-configuration [29], embedded servers [29], microservices support [30], RESTful APIs [31], and Spring Security [32] for robust security measures and testing support [33]. The backend architecture follows a layered design [34], including a presentation layer for managing API endpoints, a service layer for business logic, and a data access layer using Spring Data JPA [35] for database interactions where developers can use it to execute CRUD activities and intricate queries with minimum boilerplate code [36].

The Timefold Solver [22] serves as the central component of the timetable generation method, utilizing its advanced capabilities to efficiently distribute resources and time slots. Timefold is the subsequent development of OptaPlanner [21]. It is a type of mathematical optimization that is used in the fields of Operations Research and Artificial Intelligence. Timefold Solver is an artificial intelligence planning solver implemented in the Java programming language.

Timefold enhances the efficiency of planning problems, such as the vehicle routing problem (VRP) [37], maintenance scheduling [38], job shop scheduling [39], and school timetabling [1]. It develops logistics strategies that significantly minimize expenses, enhance the quality of service, and reduce the environmental impact, frequently by up to 25%, for intricate, practical scheduling activities. It allows for the inclusion of constraints written in code. The Timefold Solver utilizes Plain Old Java Objects (POJOs) to implement the input and output data, which consist of the planning challenge and the optimal solution. The Timefold Solver provides three types of optimization algorithms for generating high-quality solutions, namely, Construction Heuristics [40], Exhaustive Search [41], and Metaheuristics [42].

It handles hard constraints, medium constraints, and soft constraints [43]. The optimization process (or Local Search Phase [44]) uses algorithms like Hill Climbing [45], Tabu Search [46], and Simulated Annealing [47] for further refinement of scheduling solutions.

3.2. Problem Formulation

In the context of generating a university timetable using the Timefold Solver, the objective function is designed to satisfy multiple constraints and to find an optimal or near-optimal assignment of lessons to rooms and timeslots. This involves assigning each lesson to a specific room and timeslot while minimizing violations of the constraints.

Given that the timetable generation problem is a Constraint Satisfaction Problem (CSP) [48], the objective function can be structured as a multi-objective optimization function, where each constraint is weighted according to its importance.

Let,

- $L = \{l_1, l_2, \ldots, l_n\}$ represent the set of lessons that need to be scheduled.
- $R = \{r_1, r_2, \ldots, r_m\}$ represent the set of rooms in which lessons can take place.
- $T = \{t_1, t_2, \ldots, t_p\}$ represent the set of discrete time intervals during which lessons can be scheduled.
- $S = \{s_1, s_2, \ldots, s_q\}$ be the set of student groups.
- $D = \{d_1, d_2, \ldots, d_5\}$ represent the set of days in the week.
- $cap\ (r)$ be the capacity of room r.
- $students\ (l)$ be the number of students enrolled in lesson l.
- $teacher\ (l)$ be the teacher assigned to lesson l.
- $duration\ (l)$ be the duration of lesson l.
- $duration\ (t)$ be the duration of timeslot t.
- $group\ (l)$ be the student group attending lesson l.
- $room\ (l)$ denote the room assigned to lesson l.
- $building\ (r)$ represent the building associated with room r.
- C_{hard}, C_{medium}, C_{soft} represent the hard, medium, and soft constraints, respectively.

The objective function can be written as follows:

$$Z = \sum_{h \in C_{hard}} w_h \cdot V_h + \sum_{m \in C_{medium}} w_m \cdot V_m - \sum_{s \in C_{soft}} w_s \cdot V_s \quad (1)$$

where

- Z is the overall cost (objective value) to be minimized.
- w_h, w_m, w_s are weights associated with the hard, medium, and soft constraints, respectively (with $w_h \gg w_m \gg w_s$, since hard constraints are the most critical).
- V_h, V_m, V_s represent the number of violations for the respective constraint types (hard, medium, and soft, respectively).
- The objective is to minimize Z, ideally achieving $V_h = 0$ to ensure a feasible timetable, while minimizing V_m and maximizing V_s. In practice, this function is represented as a 3-level score in the system in the following way: (V_h hard/V_m medium/V_s soft).

Let $x_{l,r,t}$ be a binary decision variable, where

$$x_{l,r,t} = \begin{cases} 1, & if\ lesson\ l\ is\ assigned\ to\ room\ r\ at\ timeslot\ t, \\ 0, & otherwise \end{cases} \quad (2)$$

Thus, the decision space for the problem can be represented as a 3D matrix $X \in \{0,1\}^{n \times m \times p}$, where each element $x_{l,r,t}$ indicates whether a lesson l is assigned to a room r and a timeslot t, and the search space size, where the possible number of solutions for the timetable problem can be determined, is $S = (p \times m)^n$.

Constraints

Developing an efficient schedule for a faculty requires careful consideration of several constraints [49], as they are essential in ensuring that the timetable satisfies the requirements and preferences of students, teachers, and the institution as a whole.

Hard constraints are rigid regulations that must be followed precisely. These constraints are important for preserving the essential structure and feasibility of the timetable. Breaking hard constraints would result in infeasible timetables.

(a) Courses must be organized according to the specified time slots provided by the faculty, ensuring the timing is suitable. Let *allowed* (l,t) be a binary indicator function that specifies whether timeslot t is allowed for lesson l:

$$x_{l,r,t} \cdot (1 - allowed(l,t)) = 0 \ \forall l \in L,\ r \in R,\ t \in T \quad (3)$$

(b) The room capacity should be sufficient to accommodate the number of students enrolled in each lesson:

$$x_{l,r,t} \cdot students(l) \leq cap(r) \ \forall l \in L,\ r \in R,\ t \in T \quad (4)$$

(c) A teacher is unable to simultaneously teach two distinct lessons in separate classrooms. For any two lessons l_1 and l_2 that share the same teacher *teacher* (l_1) = *teacher* (l_2), they cannot be scheduled in the same timeslot:

$$x_{l1,r1,t} + x_{l2,r2,t} \leq 1 \ \forall\ l_1, l_2 \in L\ with\ teacher(l_1) = teacher(l_2),\ r_1, r_2 \in R,\ t\ \epsilon\ T \quad (5)$$

(d) It is not possible for a room to accommodate two distinct lessons simultaneously. For any two lessons l_1 and l_2, they cannot both be scheduled in the same room r at the same timeslot t:

$$x_{l1,r,t} + x_{l2,r,t} \leq 1 \ \forall\ l_1, l_2 \in L,\ l_1 \neq l_2, r \in R, t \in T \quad (6)$$

(e) The scheduling of lessons should not have any overlapping:

$$\sum_{r \in R} \sum_{t \in T} x_{l,r,t} = 1 \forall l \in L \tag{7}$$

(f) A lesson with a duration of x cannot be scheduled for a time slot with a different duration than x:

$$x_{l,r,t} \cdot (duration(l) - duration(t)) = 0 \; \forall l \in L, \; r \in R, \; t \in T \tag{8}$$

(g) A student group cannot attend two different lessons at the same time. For any two lessons l_1 and l_2 attended by the same student group $group\ (l_1) = group\ (l_2)$, they cannot be scheduled in the same timeslot:

$$x_{l1,r1,t} + x_{l2,r2,t} \leq 1 \; \forall \; l_1, l_2 \in L \; with \; group(l_1) = group(l_2), \; r_1, r_2 \in R, \; t \; \epsilon \; T \tag{9}$$

Medium constraints are positioned between hard and soft constraints in terms of their level of flexibility and relevance. While medium constraints offer greater flexibility compared to hard constraints, it is essential to adhere to them to generate a timetable that can be both useful and efficient.

(a) Courses and seminars should be organized together in the same time slot and room, either for a whole series of students (in the case of courses) or for a specific group of students (in the case of seminars):

$$x_{c,r,t} = x_{s,r,t} \; \forall \; c \in L_{course}, \; s \in \; L_{seminar}, \; r \in R, \; t \in T \tag{10}$$

(b) The maximum number of hours of courses per day for a student group should not exceed 10 h. Let h_d (s) be the total number of hours of courses scheduled for student group s on day d:

$$h_d(s) \leq 10 \; \forall s \in S, \; d \in D \tag{11}$$

(c) A teacher's workload should not exceed 12 h of courses each day. Let h_d (*teacher*) be the total teaching hours assigned to teacher *teacher* (l) on day d:

$$h_d(teacher) \leq 12 \; \forall teacher(l), \; d \in D \tag{12}$$

(d) Courses should be arranged based on the availability of teachers during specific time slots for teaching. Let *avail* (t, *teacher* (l)) be the availability function of teacher *teacher* (l), where 1 indicates the teacher is available during timeslot t and 0 otherwise. A lesson can only be scheduled if the teacher is available at that timeslot.

$$x_{l,r,t} \cdot (1 - avail(t, teacher(l))) = 0 \; \forall l \in L, \; r \in R, \; t \in T \tag{13}$$

Soft restrictions are preferences or recommendations that are desired to be met but can be eased if needed. These limits enhance the overall quality and satisfaction of the timetable but are not essential to its fundamental practicality.

(a) Teachers prefer instructing in a singular room to minimize the necessity of moving between several rooms. Let $\delta\left(teacher(l_i), \; teacher\left(l_j\right)\right)$ be 1 if the same teacher is assigned to both lessons l_i and l_j and 0 otherwise, and let t_i, t_j represent consecutive timeslots:

$$\sum_{i \neq j,} \delta\left(teacher(l_i), teacher\left(l_j\right)\right) \times \left(room(l_i) \neq \; room\left(l_j\right)\right), \; \forall \; l_i, l_j \in L, \; t_i < t_j, \; teacher(l_i) = teacher\left(l_j\right), \; d_i = \; d_j \tag{14}$$

(b) Students prefer to have their courses located in the same building to reduce travel time and effort between lessons. Let $\delta\left(group(l_i), \; group\left(l_j\right)\right)$ be 1 if the same student group

is attending both lessons l_i and l_j and 0 otherwise, and let t_i, t_j represent consecutive timeslots:

$$\sum_{i \neq j,} \delta(group(l_i), group(l_j)) \times (building(room(l_i)) \neq building(room(l_j))), \forall\, l_i, l_j \in L,\ t_i < t_j,\ group(l_i) = group(l_j),\ d_i = d_j \quad (15)$$

(c) Gaps between consecutive lessons should not exceed four hours, as long breaks can disrupt students' schedules. Let *start* (t) represent the start time of timeslot t, and *end* (t) represent the end time.

$$\sum_{i \neq j,} \delta\left(group(l_i), group\left(l_j\right)\right) \times max\left(0,\ start\left(t_j\right) - end(t_i) - 4\right),\ \forall\, l_i, l_j \in L,\ t_i < t_j,\ group(l_i) = group\left(l_j\right),\ d_i = d_j \quad (16)$$

(d) It is advisable to arrange laboratory courses at the same time and in the same room for student groups to ensure consistency. Let *isLab* (l) be 1 if lesson l is a laboratory lesson and 0 otherwise.

$$\sum_{i \neq j,} \delta\left(group(l_i), group\left(l_j\right)\right) \times \times \left(room(l_i) \neq room\left(l_j\right)\right) \times isLab(l_i) \times isLab\left(l_j\right),\ \forall\, l_i, l_j \in L,\ group(l_i) = group\left(l_j\right) \quad (17)$$

(e) Teachers prioritize maintaining a continuous and efficient teaching schedule by avoiding gaps between their classes. Let t_i, t_j represent consecutive timeslots:

$$\sum_{i \neq j,} \delta\left(teacher(l_i), teacher\left(l_j\right)\right) \times \left(start\left(t_j\right) - end(t_i)\right),\ \forall\, l_i, l_j \in L,\ t_i < t_j,\ teacher(l_i) = teacher\left(l_j\right),\ d_i = d_j \quad (18)$$

3.3. Database Considerations

The storage of pertinent information, including course details, student inclinations, faculty accessibility, and timetable arrangements, mandates the utilization of a database. The recommended database solution for the proposed system architecture, which consists of an Angular-based frontend and a Java-based backend powered by Spring Boot, is a Relational Database Management System (RDBMS). An option that is suitable and used in the implementation of this scenario is PostgreSQL [50].

PostgreSQL is a resilient and feature-rich open-source relational database management system (RDBMS). The system exhibits robust data integrity and concurrency control mechanisms, rendering it a fitting solution for managing concurrent requests originating from both the Angular frontend and Java backend.

PostgreSQL's ACID [51] compliance ensures transactional integrity, which is crucial for maintaining reliable and accurate data in a timetabling system. Its focus on data integrity [51] guarantees that information remains accurate and consistent, even when dealing with complex datasets and relationships. Furthermore, PostgreSQL's efficient concurrency control mechanisms [52] allow for the smooth handling of multiple users interacting with the database simultaneously, contributing to a seamless and reliable user experience.

3.4. Integration and Communication

The integration and communication aspects are an essential element. To facilitate seamless communication between the frontend and backend, the system employs RESTful APIs. The backend system provides a series of clearly defined endpoints that the Angular frontend can utilize to retrieve and submit data. The exchange of information between the frontend and backend relies on the HTTP protocol, thereby guaranteeing the ability of various system components to work together and function properly. A workflow of the application is presented in Figure 2.

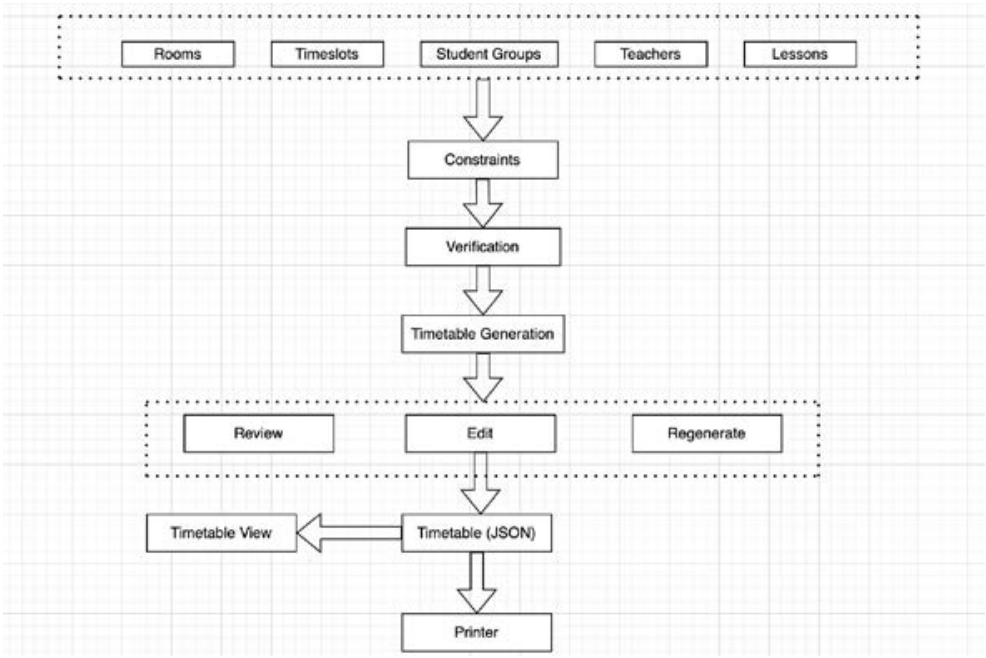

Figure 2. Proposed workflow architecture.

3.5. Enhancing System Integrity

The topic of concern in this subsection is the protection of privacy and data security.

Preserving data security and privacy is essential in any academic setting. The system must incorporate suitable measures to safeguard confidential information, including but not limited to student records and faculty data. This entails the deployment of robust authentication and authorization protocols, encryption of data both in transit and at rest, and strict adherence to established data privacy standards.

In the ongoing pursuit of fortifying the "Automatic Timetable Generator for Faculty" project, the paramount focus extends beyond the foundational aspects of data security and privacy. This section delves into the meticulous considerations and implementation strategies to ensure not only the safeguarding of sensitive information but also the reliability, security, and availability of the entire system.

3.5.1. Data Security and Privacy

A key component of this academic-focused system's operation and design is the preservation of data confidentiality and privacy. The system's purview encompasses safeguarding confidential information, ranging from student records to faculty data. To fortify the integrity of the data, it is required several measures. Robust authentication and authorization protocols restrict access to authorized individuals based on their roles within the academic hierarchy. Hashing techniques convert sensitive data into irreversible strings of characters, adding a layer of protection against unauthorized access [53]. The system also adheres to established data privacy standards, ensuring legal and ethical handling of personal data.

3.5.2. Reliability: The Timefold Solver Library

Reliability is a paramount aspect of the system, especially in the context of timetable generation. To achieve unparalleled reliability, the project leverages the robust capabilities of the Timefold Solver library. The library ensures this by using advanced algorithms rigorously tested with various datasets. These algorithms generate accurate, conflict-free timetables, contributing to overall system reliability. The library also includes error detection and handling mechanisms, allowing it to quickly identify and mitigate issues, preventing disruptions.

3.5.3. Security: Role Guards and JWT Authentication

Security is important in safeguarding the system against unauthorized access and potential breaches. Security is enhanced through role guards [54], which ensure only users with specific permissions can access parts of the application. This protects critical information and prevents unauthorized tampering. The system also employs JSON Web Token (JWT) [55] authentication to securely verify user identities. This, combined with strong authorization mechanisms, ensures that only authenticated and authorized users can interact with the system, reducing the risk of security breaches.

4. Research Methodology

To address the research objectives and develop an effective solution for automatic timetable generation in a faculty or college, a structured research methodology will be followed. This section outlines the methodology that guides the research process, including the steps involved in data collection, system design, implementation, and evaluation.

4.1. Problem Analysis and Requirements Gathering

The research begins with a thorough analysis of the existing timetabling process and the specific challenges faced by the faculty or college. This involves collecting relevant information from faculty administrators, staff members, and students. The objective is to identify the key requirements, constraints, and preferences that need to be considered in the automated timetable generation system.

4.2. Literature Review

An extensive literature analysis will be performed to evaluate existing approaches, algorithms, and techniques used in automatic timetable generation. This involves studying research papers, conference proceedings, and relevant publications in the field of timetabling and optimization. The literature review provides a solid foundation for understanding the state-of-the-art methods and identifying potential gaps in the existing solutions, as presented in the Related Studies section.

4.3. System Design

Based on the requirements gathered and the insights gained from the literature review, the research focuses on designing the system architecture, algorithms, and data models for automatic timetable generation. This involves identifying the appropriate algorithms and techniques that can effectively address the specific challenges of the faculty or college. The system design aims to optimize resource allocation, minimize conflicts, and satisfy the various constraints and preferences identified in the problem analysis.

4.4. System Implementation

Once the system design is finalized, the research proceeds with the implementation phase. The web platform is developed using Angular for the frontend and Java with Spring Boot for the backend, as mentioned earlier. The algorithms and techniques selected during the system design phase will be implemented, and the necessary APIs and database connections will then be established. This phase also involves testing and debugging to ensure the correctness and functionality of the system.

4.5. Data Collection and Integration

To generate effective timetables, relevant data will be collected and integrated into the system. This may include course information, faculty availability, student/teacher preferences, and any additional constraints specific to the faculty or college. Data collection can involve surveys, interviews, or data extraction from existing systems. The collected data will be processed and integrated into the system to provide accurate inputs for the timetable generation process.

4.6. Evaluation and Performance Analysis

Once the system is implemented and the data are integrated, an evaluation process will be conducted to assess the performance and effectiveness of the automated timetable generation system. This involves generating timetables based on different scenarios and real-world data and comparing the results against predefined metrics and objectives. The evaluation focuses on factors such as timetable quality, resource utilization, and user satisfaction. Performance analysis will also be conducted to measure the system's responsiveness, scalability, and efficiency.

4.7. Iterative Refinement

Based on the evaluation results, feedback from stakeholders, and insights gained during the implementation and evaluation phases, the system will undergo iterative refinements [56]. This may involve fine-tuning the algorithms, enhancing the user interface, and addressing any identified limitations or shortcomings. The iterative refinement process will continue until a satisfactory level of performance and user acceptance is achieved.

4.8. Ethical Considerations

Throughout the research process, ethical considerations will be given utmost importance. The privacy and security of student and faculty data will be ensured, and informed consent will be obtained for data collection. Any potential biases or unintended consequences of the automated timetable generation system will be carefully analyzed and mitigated. The research will adhere to ethical guidelines and regulations to maintain the integrity and welfare of the participants.

5. The Development of Theoretical/Practical Contributions

The development of an "Automatic Timetable Generator for Faculty" represents a pivotal advancement in the realm of academic management systems. This section delves into the intricate details of the application's implementation, elucidating the theoretical underpinnings and practical functionalities that characterize this innovative tool.

5.1. Comprehensive Approach to Automated Timetable Generation

The process of developing theoretical and practical contributions for automated schedule production commenced with a comprehensive review of the present timetabling method at University Politehnica Bucharest. This entailed in-depth consultations with the faculty member in charge of manual timetable creation, along with thorough literature analysis and system design.

5.1.1. Problem Analysis

During the initial consultations with the faculty member, important features of the timetabling procedure were identified, as can be seen in Table 1.

Table 1. Timetabling procedure's features.

Feature	Description
Student hierarchy	Students are categorized into series, which are then separated into groups, and these groupings can be further divided into semigroups. Classes are conducted for complete series, seminars for groups, and laboratory sessions for either groups or subgroups.
Room and time constraints	Lessons must be assigned to rooms that have suitable capacity to handle the student groups. In addition, students are limited to a maximum of 10 h of coursework every day.

Table 1. *Cont.*

Feature	Description
Scheduling complexity	The interplay of these constraints leads to a high level of complexity in scheduling, requiring the allocation of many lessons into available rooms and timeslots without any clashes. The timetable generation problem includes 153 rooms with different capacities and located in different buildings, 195 timeslots (from Monday to Friday), 18 constraints of hard, medium, or soft levels, 166 teachers, 4187 lessons, and 338 students' groups (including Bachelor and Master students). The schedule is made on a weekly basis during the semester. This is the highest complexity, compared with other papers, research articles, and implementations.

5.1.2. Literature Overview

An extensive literature assessment was undertaken to investigate the strategies, algorithms, and techniques employed in the automated generation of timetables. In the areas of scheduling and optimization, research papers and related publications were reviewed. Although there have been several studies on evolutionary algorithms, constraint satisfaction, and heuristic approaches, there is a clear lack of research addressing the specific difficulties of scheduling university timetables with a high volume of classes, as observed at University Politehnica Bucharest.

5.1.3. Data Collection and Integration

Data related to lessons, student groups, rooms, teachers, timeslots, and limits was gathered from the online timetable accessible on the Politehnica University of Bucharest website. The data underwent processing and integration into the system to ensure precise inputs (Table 2) for the development of the timetable.

Table 2. Timetable inputs.

Input	Description
Lessons	Detailed information about courses, seminars, and laboratory sessions.
Student groups	Organization of series, groups, and semigroups.
Rooms	Name, capacity, and location details.
Teachers	Name and availability.
Timeslots	Defined periods for scheduling lessons.
Constraints	Rules for maximum hours per day, room capacity, lesson type allocation, and the basic constraints that exist in the university/college timetable.

5.1.4. Ethical Considerations

Given that the data utilized for timetable construction were publicly available online, ethical considerations were not a major consideration in this research. Therefore, matters concerning privacy and informed consent were not applicable.

5.1.5. Database Model for Timetable Generation

An effective and organized database model is essential for the efficient creation of timetables. The database should be structured to accommodate diverse entities, including student groups, teachers, classes, rooms, limitations, time slots, and users (both administrators and regular users).

The primary entities in the database model (Table 3) consist of *StudentGroup*, *Teacher*, *Lesson*, *Room*, *ConstraintModel*, *Timeslot*, and *User*. Every entity possesses distinct attributes and establishes connections with other entities, which are essential for the effective operation of the timetable generation system.

Table 3. Database model entities.

Entity	Attributes	Relationships
StudentGroup	- Id (Primary Key): Unique identifier for each student group. - Year: year of the student group - Name: Name of the series of student group (e.g., AA1 and AB3). - StudentGroup (group): Group to which the students belong. - SemiGroup: Semigroup to which the students belong (Semigroup 1, Semigroup 2, or Semigroup 0 for the Master students). - NumberOfStudents: Number of students in the group.	- A student group can be linked to multiple lessons. - A student group is associated with specific constraints.
Teacher	- Id (Primary Key): Unique identifier for each teacher. - Name: Full name of the teacher. - Timeslots: Time slots during which the teacher is available to teach.	- A student group can be linked to multiple lessons. - A teacher is associated with specific constraints.
Lesson	- Id (Primary Key): Unique identifier for each lesson. - Subject: Lesson's name - TeacherId (Foreign Key): Identifier of the teacher conducting the lesson. - StudentGroupId (Foreign Key): Identifier of the student group attending the lesson. - LessonType: Type of lesson (e.g., Course, Seminar, and Laboratory). - Year: The year for which the lesson is held. - Duration: Duration of the lesson. - TimeslotId (Foreign Key): Identifier of the timeslot where the lesson is assigned. - RoomId (Foreign Key): Identifier of the room where the lesson is held.	- A lesson is linked to a specific teacher and student group. - A lesson must be scheduled in an appropriate room and time slot.
Room	- Id (Primary Key): Unique identifier for each room. - Name: Name or number of the room. - Capacity: Maximum number of students the room can accommodate. - Building: Physical location of the room.	- A room can host multiple lessons. - A room is associated with specific constraints.
ConstraintModel	- Id (Primary Key): Unique identifier for each constraint. - Description: Description of the constraint. - Weight: Type of constraint (e.g., Hard, Medium, and Soft).	- Constraints are linked to student groups, teachers, rooms, and lessons. - Constraints must be considered during timetable generation to ensure compliance.
Timeslot	- Id (Primary Key): Unique identifier for each time slot. - Day: Day of the week (e.g., Monday and Tuesday). - StartTime: Start time of the time slot. - EndTime: End time of the time slot.	- A time slot can be assigned to multiple lessons. - Time slots must align with teacher availability.
User	- Id (Primary Key): Unique identifier for each user. - Email: Contact email of the user. - Password: Hashed password for the user account. - Role: Role of the user (e.g., Admin and Normal User).	- Users can manage and access timetable data according to their role. - Admin users have additional permissions for configuring constraints and settings or for generating the timetable.

To better understand the relationships between these entities, an Entity-Relationship (ER) diagram can be constructed, as seen in Figure 3.

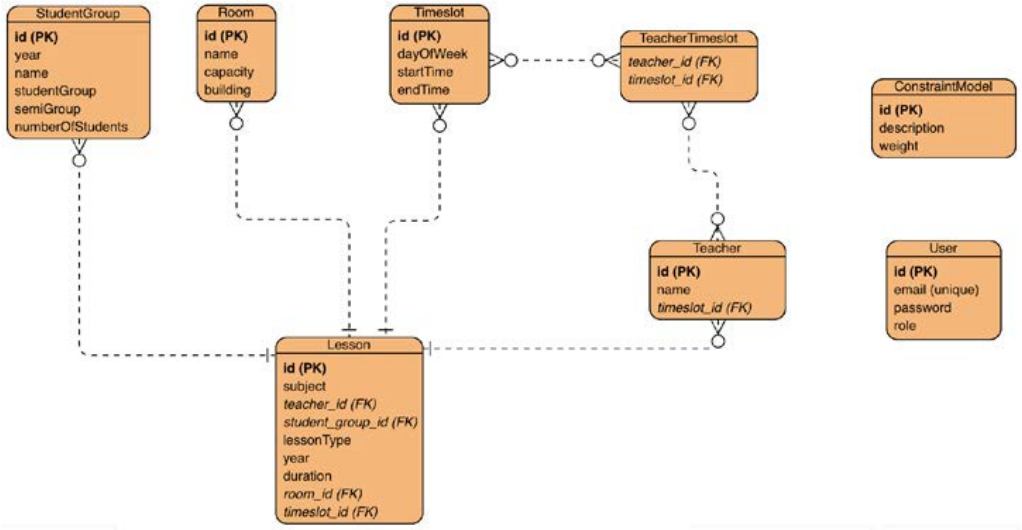

Figure 3. ER diagram.

5.2. Functionalities of the Application

In the pursuit of optimizing academic scheduling, an effective Automatic Timetable Generator for Faculty encapsulates multifaceted functionalities. The application allows administrators to define and manage courses and various academic activities such as seminars, laboratories, and exams. It provides options to specify activity durations, resource requirements, and any constraints associated with each activity. Furthermore, the application supports the management of faculty members and staff by allowing administrators to input their availability, preferences, and workload constraints. This data are essential for generating optimized timetables that take individual preferences into account.

Additionally, the application facilitates the management of rooms and resources available within the faculty. It allows administrators to define room capacities, availability, and any specific requirements for activities, ensuring that appropriate rooms and resources are allocated during the timetable generation process. The core functionality of the application is to generate automated timetables based on the provided input data. To achieve this, it employs optimization algorithms, such as Tabu Search algorithms or other suitable techniques, to create conflict-free and optimized timetables while considering various constraints and preferences.

Moreover, the application has mechanisms to identify and resolve conflicts that may arise during the timetable generation process. Conflicts include overlapping activities, resource constraints, faculty availability issues, or student preferences that cannot be accommodated. The system provides options for resolving these conflicts based on predefined rules or user-defined priorities. Finally, the generated timetables are presented in a user-friendly and intuitive manner. The application provides visualization tools to display timetables, allowing users to easily view and navigate through the schedules.

5.3. Use Case UML Diagram

The Use Case Diagram in UML stands as a visual representation of the functional requirements and interactions within the system, offering a comprehensive view from a user's perspective. It identifies the different actors (users, systems, and external entities) and the various use cases (functionalities or actions) they perform within the system. It is noteworthy that there are three distinct categories of users within the system: administrators, teachers, and students. The administrator is responsible for constructing the schedule by accessing the designated platform, inputting pertinent data to facilitate schedule generation, making necessary modifications, and reviewing the resultant timetable. Conversely,

students and teachers are limited to viewing the schedule without the ability to make changes. The use case diagram can be found in Figure 4.

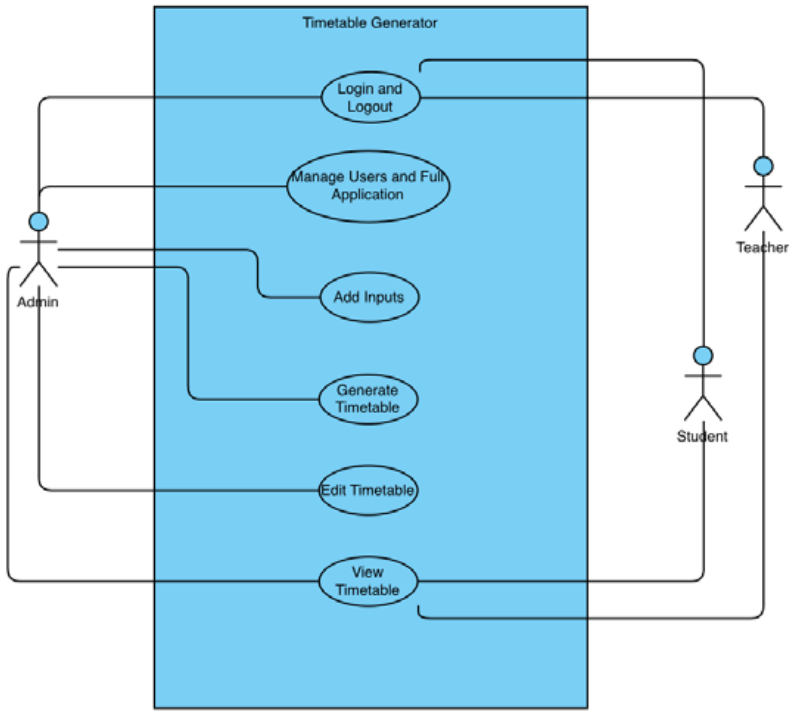

Figure 4. Use case diagram.

5.4. Implementation Details

5.4.1. User Authentication and Authorization

The key component of the application is based on the meticulous implementation of user authorization and authentication, with a focus on ensuring data security and controlling access. Utilizing the BCrypt hashing algorithm for password storage is a good choice for protecting user credentials. The adaptive hashing technique used by BCrypt, along with the addition of salting, effectively reduces the probability of password-related vulnerabilities. Every user in the database has a password that has been hashed and salted, which provides a safe basis for user authentication, as demonstrated in Figure 5.

	id [PK] integer	email character varying (255)	password character varying (255)	role character varying (20)
1	1	admin@gmail.com	$2a$12$ySaIIfWLimnqVTf/.dIWRuwpVhsjBO1nV3fNTKolfpgLhidD6Hp...	ADMIN
2	2	user@gmail.com	$2a$12$ySaIIfWLimnqVTf/.dIWRuwpVhsjBO1nV3fNTKolfpgLhidD6Hp...	USER

Figure 5. Users' database with hashed password.

Upon initiating the login process, the Spring Security module orchestrates a sophisticated authentication mechanism. A comprehensive security filter chain evaluates user credentials against the stored data in the database. Successful authentication triggers a seamless redirection to the application dashboard, where the user's assigned role—either ADMIN or USER—determines the extent of their access privileges. The authentication process extends to the generation of a JWT token (Figure 6), providing a time-limited, secure session for users.

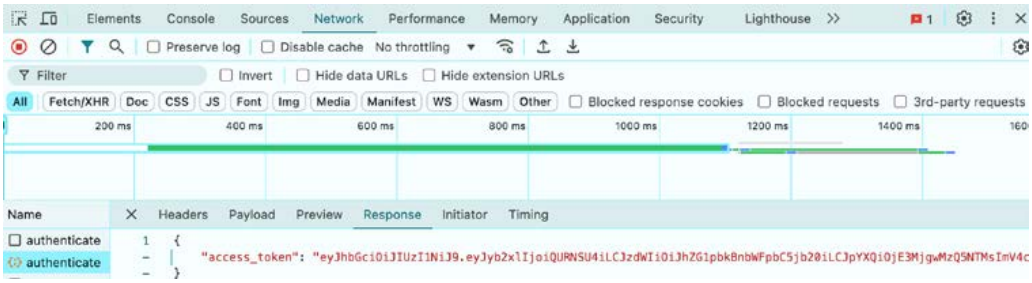

Figure 6. Successful authentication with JWT token response.

5.4.2. CRUD Operations with Spring JPA

The dynamic nature of the application necessitates the seamless handling of Create, Read, Update, and Delete (CRUD) operations. Leveraging the capabilities of Spring JPA, the system adeptly manages essential data entities required for timetable generation. These entities encompass teachers, students, classrooms, courses, and restrictions, among others. The implementation ensures a streamlined and extensible approach to handling the core data components that fuel the timetable generation process.

Angular guards play a pivotal role in maintaining a secure and role-based access environment. These guards thoroughly examine user permissions and roles, ensuring that only authorized users are granted the ability to access the specific pages. Unauthorized attempts to access restricted pages result in automatic redirection to the login page, thereby enhancing the overall security posture of the application.

5.4.3. Timefold for Timetable Generation

The main feature of the application is centered around the advanced integration of Timefold for generating timetables. Timefold, a powerful constraint satisfaction solver, simplifies the difficult task of generating university timetables. The approach utilizes optimization algorithms, effectively integrating Java annotations and configurations with Spring Boot [57], as can be seen in Table 4.

Table 4. Timefold annotations and configurations.

Annotation	Description
@PlanningEntity	Marks classes as entities manipulated by Timefold to identify an optimal solution. Entities include activities, teachers, and classrooms.
@PlanningVariable	Identifies variables subject to change during optimization, such as time slots and rooms for each activity.
@PlanningSolution	Designates the class representing the entire problem to solve, with the timetable serving as the planning solution.
@SolverFactory	Configures and initializes the solver, allowing for custom settings such as optimization algorithms and termination conditions.
@PlanningScore	Identifies the attribute representing the score or fitness of the solution, guiding Timefold in refining the timetable.
@ConstraintConfigurationProvider	Holds the configured constraint weights and other constraint parameters.
@PlanningEntityCollectionProperty	Timefold Solver can change the annotated object during solving, and the ConstraintProvider (where the constraints are implemented) can select from those too.
@ProblemFactCollectionProperty	The ConstraintProvider can select from those annotated object instances to assign to a @PlanningEntity object.

The integration with Timefold represents an important achievement in the application, providing a powerful and adaptive solution for efficient university timetable generation. Figure 7 represents the utilized domain model for the timetable generation with Timefold annotations.

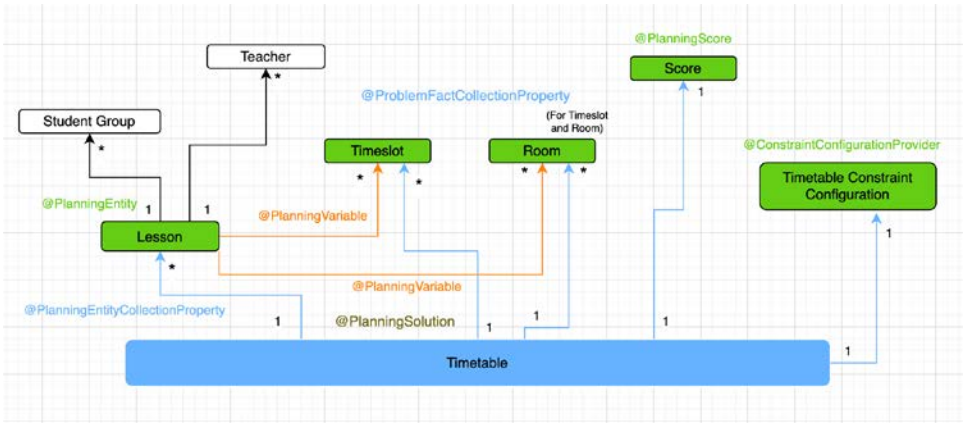

Figure 7. Timefold domain model. The * means "multi" relationship.

In the development of the automatic timetable generation system, the following Timefold Solver configuration (Table 5) was used to ensure efficient and optimized scheduling.

Table 5. Timefold configuration.

	Timefold Configuration	
	InitializingScoreTrend	**MoveThreadCount:**
Description	This parameter defines the trend of the score during the initialization phase, while the previously initialized variables remain unchanged. Some optimization algorithms, including Construction Heuristics, can achieve better execution times when provided with such information.	This parameter is set to AUTO, which allows Timefold to automatically determine the optimal number of threads for performing move evaluations in parallel. This enhances the solver's performance by leveraging multi-core processors, thereby reducing the computational time required for generating optimized timetables.

The configuration is followed by the metaheuristics algorithms that occur in two phases: Construction Heuristics and Local Search. The Construction Heuristics phase is creating the best feasible initial using the Strongest Fit Decreasing algorithm that focuses on the most challenging aspects of the timetable, thereby laying a strong foundation for further optimization. Then, to refine the initial solution and avoid getting stuck in local optima, two advanced local search algorithms were employed: Hill Climbing Late Acceptance and Tabu Search.

The Strongest Fit Decreasing (SFD) algorithm operates by first sorting lessons, timeslots, and rooms based on specific criteria and then assigning the most "difficult" lessons to the most suitable rooms and timeslots. Mathematically, this can be described as follows: Each lesson l_i has an associated difficulty or strength value d (l_i) based on the duration of the lesson. Each timeslot t_j has an associated importance score imp (t_j), such as the interval between start and end times, and each room r_k has a capacity value C (r_k). Sort the lessons L, timeslots T, and rooms R such that $d(l_1) \geq d(l_2) \geq \ldots \geq d(l_n)$, $imp(t_1) \geq imp(t_2) \geq \ldots \geq imp(t_p)$ *and* $C(r_1) \geq C(r_2) \geq \ldots \geq C(r_m)$. For each lesson l_i (in descending order of difficulty), find the most suitable pair of room r_k and timeslot t_j that

minimizes constraint violations. This can be formulated as minimizing the cost function $f(l_i, r_k, t_j)$, which represents the penalty for assigning a lesson l_i to room r_k in timeslot t_j:

$$\min_{l_i \in L, r_k \in R, t_j \in T} f(l_i, r_k, t_j) \tag{19}$$

By assigning the most difficult lessons first to the best available room-timeslot pairs, the algorithm tackles the hardest parts of the problem early, reducing the complexity of the remaining assignments.

The Late Acceptance Algorithm is a local search algorithm that accepts solutions based on their quality relative to past solutions rather than the immediate current solution. This approach helps avoid being trapped in local optima. This algorithm can be described as follows: Let *LA_size* be the number of previous solutions to be considered and *accepted-CountLimit* be the number of consecutive non-improving moves allowed before stopping. Let the initial solution be S_0 with an objective value $Z(S_0)$. Maintain a memory buffer $M = [Z(S_1), Z(S_2), \ldots, Z(S_{LA\_size})]$ of the objective values of the last *LA_size* solutions. For each iteration k, generate a new candidate solution S_k and calculate its objective value $Z(S_k)$. The new solution S_k is accepted if:

$$Z(S_k) < Z(S_{k-LA\_size}) \tag{20}$$

That is, the new solution is accepted if it is better than the solution from *LA_size* iterations ago, even if it is worse than the current solution. If no better solution is found after *acceptedCountLimit* consecutive iterations, the search is terminated.

Tabu Search is a local search algorithm that uses memory structures to avoid revisiting recently explored solutions, encouraging the search to explore new regions of the solution space. Mathematically, it can be described as follows: Let *Tabu_size*, the size of the tabu list (how many recent solutions are considered "tabu") and *acceptedCountLimit*, the number of consecutive iterations allowed before termination. Start with the initial solution be S_0 with an objective value $Z(S_0)$. In each iteration, explore the neighborhood $N(S_k)$. of the current solution S_k.

The neighborhood $N(S_k)$ consists of solutions that can be reached by making small changes to S_k (e.g., swapping rooms or timeslots for two lessons). Maintain a tabu list T of size *Tabu_size*. A move that leads to a solution that is in the tabu list is forbidden (tabu) unless it satisfies an aspiration criterion (i.e., if it significantly improves the solution). If the move leads to a solution S_{k+1} such that $Z(S_{k+1}) < Z(S_{best})$, it is accepted regardless of whether it is tabu. Among the non-tabu moves in the neighborhood $N(S_k)$, select the move that leads to the solution with the best objective value $Z(S_{k+1})$. The new solution S_{k+1} is added to the tabu list, and the oldest entry in the tabu list is removed, keeping the tabu list size constant at *Tabu_size*. The search is terminated if no improvement is found after *acccceptedCountLimit* iterations.

These algorithms work together to ensure that the timetable generation process is both efficient and effective for real-world scenarios, producing high-quality solutions that meet the diverse constraints and preferences of the stakeholders.

Timefold constraints are implemented by creating a network composed of nodes. The nodes representing the sources are the *forEach()* nodes, while the nodes representing the sinks are the penalties. The solver sequentially traverses all problem facts and entities across this graph according to a predetermined order. The limitations are broken down into a network consisting of a single node, which leads to penalties in the conclusion. This methodology is similar to the RETE algorithm, a commonly employed method in rule-based systems for pattern matching.

The RETE algorithm, created by Forgy [58], is a very efficient algorithm for pattern matching that is used to design production rule systems. Its purpose is to efficiently compare a vast number of rules with an extensive collection of facts or data. The fundamental concept behind the RETE algorithm is to minimize the number of comparisons needed

by sharing common conditions among many rules and storing partial matches to prevent repetitive evaluations.

Upon executing the algorithm and applying the constraints, the frontend component returns a JSON-formatted response. This response is then utilized to display the data in the university timetable.

5.4.4. Security Measures: Frontend Interceptors and Backend Filters

Augmenting the security architecture, both frontend interceptors in Angular and a robust security filter in the backend using Spring Security are implemented.

Angular interceptors play a crucial role in securing communication between the frontend and backend. These interceptors facilitate the inclusion of the user's token in HTTP requests, ensuring secure and authenticated interactions. This mechanism safeguards against potential threats and unauthorized access, contributing to a resilient and secure application.

A security filter in the backend, integrated with Spring Security's *AuthenticationProvider*, analyzes each incoming request. This filter serves as the last line of defense, rigorously verifying the validity of tokens and user credentials for every request. Unauthorized attempts trigger appropriate responses, reinforcing the integrity of the system.

The combination of these security measures fortifies the application, creating a robust shield against potential security threats. The implementation ensures that data integrity, confidentiality, and availability are uncompromised, establishing a secure and dependable platform for university timetable generation.

6. Performance Analysis Measure

The evaluation of the effectiveness and efficiency of the automatic timetable generation system in a faculty or college is contingent upon the critical aspect of performance analysis. The present section centers on the utilization of performance analysis measures to evaluate diverse facets of the system's performance. Through the analysis of these metrics, valuable insights can be obtained to optimize the system and improve its overall efficiency.

6.1. Timetable Quality Metrics

The assessment of timetable quality metrics pertains to the degree to which the generated timetables conform to the stipulated requirements and constraints of the faculty or college. Several metrics may be considered. Conflict resolution evaluates the frequency of conflicts or discrepancies in the schedule, such as the occurrence of overlapping lectures or exams. A reduced frequency of conflicts is indicative of a superior-quality timetable. Timefold Solver provides a score calculation functionality that indicates the quality of the solution within a specified time.

Resource utilization refers to the measurement of the efficiency with which available resources, including but not limited to classrooms and faculty, are being utilized. This process assesses the distribution of resources to minimize instances of both underutilization and overutilization. Balanced Workload prioritizes the equitable distribution of workload among faculty members. The algorithm considers various parameters, including the number of instructional hours, intervals between courses, and successive instructional hours assigned to individual faculty members.

6.2. Computational Performance Metrics

The automatic timetable generation system is evaluated based on its efficiency and scalability through the use of computational performance metrics. The metric of execution time pertains to the duration required by the system to produce a schedule. A reduced execution time is a desirable outcome, as it signifies a more expedited timetable generation process. Scalability refers to the system's ability to maintain optimal performance levels as the scope and complexity of the problem being addressed increase. The evaluation assesses the system's capacity to process extensive datasets and produces schedules within

acceptable temporal constraints. Additionally, the evaluation of memory usage pertains to the memory demands of the system while generating the timetable, which aids in the detection of any suboptimal memory utilization or possible performance limitations.

6.2.1. User Satisfaction Surveys

The utilization of user satisfaction surveys is a valuable approach to gaining insights into the usability and user experience of the automatic timetable generation system. One potential approach to collecting feedback on the system's usability, functionality, and overall satisfaction is to administer surveys to faculty members, students, and administrators.

6.2.2. Comparison with Manual Timetabling

A comparative analysis between the automated system and the manual timetabling process can be conducted to assess the efficiency of the former. This may encompass various indicators, such as gains in efficiency, decreases in interpersonal frictions, and enhancements in the allocation of assets. The act of juxtaposing the efficiency of the mechanized system against the manual procedure serves to illustrate the merits and gains of the suggested resolution.

6.2.3. Real-World Testing and Validation

The process of real-world evaluation and verification entails the execution of the automatic timetable generation system within an authentic academic setting, such as Politehnica University of Bucharest. This facilitates the assessment of the system's efficacy in practical scenarios and facilitates the acquisition of constructive input from users. The performance of the system can be evaluated in high-demand scenarios by conducting tests during peak scheduling periods.

7. Results

The university timetable generation application was executed using two different Apple laptops, the MacBook Pro M1 (2021) and the MacBook Air M1 (2020), each with specific hardware and software configurations to evaluate the algorithm's performance and effectiveness.

The MacBook Pro M1 (2021) utilized for this study is equipped with a 10-core CPU, a 16-core GPU, 3.2 GHz, 16 GB unified memory, and a 1 TB SSD. The operating system used was macOS Monterey. The frontend was developed using Angular 16 and TypeScript 4.9.4, while the backend was implemented with Java 21, Spring Boot 3.2.3, and PostgreSQL 14.10 for the database. The solver employed was Timefold Solver 1.9.0, though the current version at the time of writing was 1.10.0.

Similarly, the MacBook Air M1 (2020) used in the experiments featured an 8-core CPU, a 7-core GPU, 3.2 GHz, 8 GB of unified memory, and a 512 GB SSD, running on macOS Monterey. The software stack for the frontend and backend was identical to that of the MacBook Pro, using Angular 16, TypeScript 4.9.4, Java 21, Spring Boot 3.2.3, and PostgreSQL 14.10 with Timefold Solver 1.9.0.

The timetable generation algorithm was tested twice on the MacBook Pro M1, each with different durations to assess performance and effectiveness. In the first run, which lasted approximately 8 h, the best score obtained was −262 init/0 hard/−2113 medium/7647 soft (Figure 8). This result indicates that there were no violations of hard constraints, 2113 violations of medium constraints, and a score of 7647 for soft constraints, with 262 rooms or timeslots unassigned to any lesson. The algorithm ran for 28,800,043 milliseconds, achieving a score calculation speed of 8401 scores per second, utilizing 4 threads for processing.

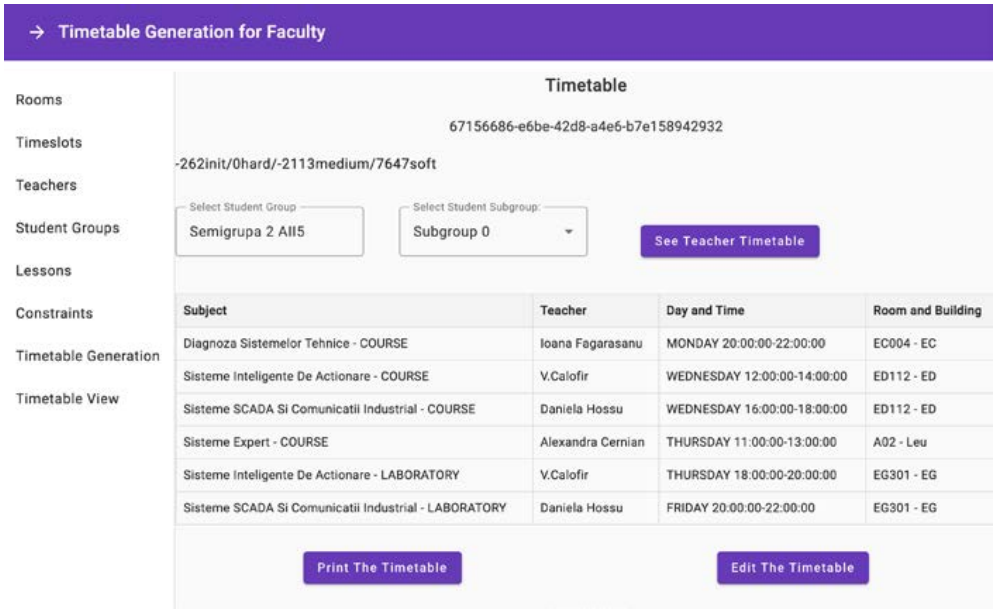

Figure 8. Results of the first run of the algorithm.

In the second run, extending to 12 h, the Construction Heuristic phase successfully initialized the best feasible solution after 8 h and 37 min with a score of 0 hard/−2187 medium/7889 soft. For the remaining hours, the Local Search phases made incremental improvements, culminating in a final score of 0 hard/−2183 medium/8121 soft (Figure 9). As with the first run, there were no violations of hard constraints, and all the rooms and timeslots were assigned to lessons, but the number of medium constraint violations decreased to 2183. The score for soft constraints improved to 8121. The algorithm ran for 43,200,005 milliseconds, with an overall score calculation speed of 6342 scores per second, and averaged 1991 scores per second during the local search phases. This run also used 4 threads for processing.

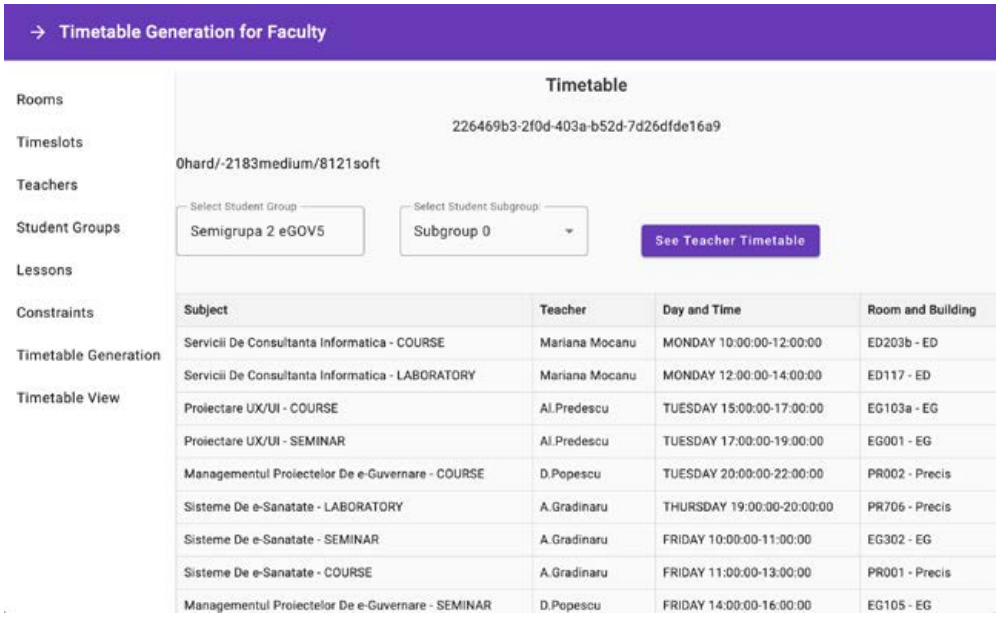

Figure 9. Results of the second run of the algorithm.

The timetable generation algorithm was also tested on a MacBook Air M1 (2020) to evaluate performance differences due to hardware variations with 4 threads for processing.

After running the algorithm for 5 h, the best score obtained was −3122 init/0 hard/−462 medium/3417 soft. Extending the runtime to 8 h resulted in a score of −1938 init/0 hard/−798 medium/5531 soft. Further extending the runtime to 11 h improved the initial unassigned count to −1102 init, with improvements in the soft constraint score but a lower medium score (−1102 init/0 hard/−1246 medium/6571 soft) compared to the 8 h run. In each case, the score calculation speed was lower in comparison to the MacBook Pro M1.

Additionally, an alternative configuration was tested on the MacBook Air M1, where timeslots were assigned to courses first, followed by room assignments during the construction heuristic phase. After 8 h, this configuration produced a score of −10 hard/−5602 medium/3361 soft. This configuration led to 10 hard constraint violations, 5602 medium constraint violations, and a soft constraint score of 3361.

Despite using 4 threads, the algorithm required considerable time to generate the timetable on the MacBook Pro M1 (Table 6). Running the algorithm for 12 h helped the solver assign lessons to classrooms and timeslots without violating any hard constraints, significantly enhancing the soft constraint score from 7629 to 8121. However, the extended runtime also increased medium constraint violations from 2102 to 2183, indicating that while the solution became more optimal regarding soft constraints, it struggled to minimize medium constraint violations effectively over the extended duration.

Table 6. Comparative execution results on different machines.

Machine	Hours	Score	Score Calculation Speed
M1 Air (I)	5	−3122 init/0 hard/−462 medium/3417 soft	8703/s
M1 Air (II)	8	−1938 init/0 hard/−798 medium/5531 soft	6666/s
M1 Air (III)	11	−1102 init/0 hard/−1246 medium/6571 soft	5478/s
M1 Air (alt. config)	8	−10 hard/−5602 medium/3361 soft	664/s
M1 Pro (I)	8	−262 init/0 hard/−2113 medium/7647 soft	8401/s
M1 Pro (II)	12	0 hard/−2183 medium/8121 soft	6342/s
Simulation (M1 Air)	6	0 hard/−1778 medium/8265 soft	73,135/s

The low medium constraint score can be related to the limited access to information from course lecturers, excluding those who teach seminars or laboratories. The assumption that these lecturers also teach the course, seminar, and laboratory was adopted. Despite imposing the constraint that a professor could not teach more than 12 h per day, the system recorded instances where a professor was allocated 13 or 14 h of lecture, resulting in scores of −1 and −2 accordingly. This also applies to student groups that are limited to 10 h of instruction per day. Furthermore, the implementation of the constraints considering teachers' preferences lacks precision. Instead of assigning estimated periods during which each teacher can teach, we assigned specific time slots. This leads to instances where a teacher is allocated to a time slot that he is unavailable for, despite being capable of teaching during that time without any difficulty.

The performance on the MacBook Air M1 revealed that shorter runtimes resulted in higher unassigned counts and lower soft constraint scores. The algorithm possibly needs at least 16 h to achieve a feasible solution with all timeslots and rooms assigned, but even then, there is no guarantee that the medium constraints violation will decrease over time. This indicates that while the algorithm can assign more lessons over time, it reaches a point where additional runtime does not significantly improve medium or soft constraint scores. Using a configuration that assigned timeslots first resulted in hard and medium

constraint violations, indicating a less effective approach for meeting all constraints. The higher medium constraint violations suggest that the initial assignment of timeslots without considering room assignments leads to suboptimal solutions.

A Spring Boot REST API simulation on a MacBook Air M1 demonstrated that generating a feasible solution with instantiated objects in Java required only 6 h. This result suggests that, while the full-stack solution involving Angular and Spring Boot provides comprehensive functionality, the backend algorithm alone can operate more efficiently when separated from the additional overhead introduced by the web application context. The simulation began with an initial score of −8320 init/0 hard/0 medium/0 soft. After approximately 15 min, the construction heuristic phase improved the score to 0 hard/−1787 medium/7848 soft. The first local search phase, completed after 2 h, further refined the score to 0 hard/−1781 medium/8248 soft. The second local search phase concluded after 3 h, achieving a final score of 0 hard/−1778 medium/8265 soft. The entire process confirmed that the backend algorithm can solve the problem efficiently when isolated from the additional overhead introduced by the web application context.

The entire process of creating a highly complex university timetable from scratch in the current state of the application would take about a week. Compared to generating a timetable from scratch in a manual way, it can be said that automation brings a big advantage to educational institutions. And in the case of schools or colleges even more so, as there is much less data, so chances are that generating a school or college timetable will only take a few days. However, the system hasn't been officially tested in the academic institution, as most of the information was extracted from the timetable available online. As a result, this application is only used on a prototype level due to the lack of provided data.

The extended run times on both the MacBook Pro M1 and MacBook Air M1 highlight the limitations of current hardware in generating optimal timetables efficiently. More powerful hardware is necessary for real-world applications to achieve quicker and more optimal results. The choice of configuration significantly impacts the quality of the generated timetable. Assigning timeslots first led to more violations, suggesting a need for integrated assignment approaches. The consistent use of four threads aligns with Timefold's recommendations, as increasing the thread count beyond this could introduce additional overhead, potentially slowing down the process due to the complexities of thread management. However, further optimization in thread management might be needed to improve performance. Unit tests written in JUnit5 validated the constraints, ensuring their correct application and enforcement during the timetable generation process.

The results demonstrate that while the algorithm can generate feasible and optimized timetables, performance and time required are significantly influenced by hardware capabilities and algorithm configurations. For real-world applications, optimizing the environment and leveraging more powerful infrastructure would be essential for handling real-world cases efficiently. Furthermore, while extended runtimes improved the solution quality regarding soft constraints, they also introduced more medium constraint violations, suggesting a need for better-written constraints and for fine-tuning the algorithm's balancing of different constraint types more efficiently.

8. Discussions and Future Studies

The evolution of automated timetable generation systems has emerged as a vital area of research, particularly in the context of educational institutions facing increasing demands for efficient scheduling solutions. While existing studies have significantly contributed to the understanding and development of these systems, they have primarily focused on smaller datasets and less complex scenarios. This study distinguishes itself by tackling the intricate challenges posed by large-scale scheduling, integrating hundreds of courses, instructors, rooms, and time slots. The ability to generate effective and efficient timetables despite these complexities is an important advancement for institutions with large student and faculty populations, as it directly impacts their operational efficiency and overall academic performance.

Several enhancements are planned to further improve the timetable generation system, ensuring it meets the diverse needs of educational institutions effectively. Users will be able to interact with the timetable, making manual adjustments or edits based on specific requirements, providing greater flexibility and customization to meet unique needs. The application will enable the generation of reports and summaries of the generated timetables, which can be used by administrators, faculty members, and students for various purposes such as academic planning, resource allocation, and schedule communication. Additionally, it will support communication features to notify stakeholders about timetable updates or changes.

Implementing load balancing [59] will enhance the system's performance, particularly when handling large datasets or multiple simultaneous users. An automated way to retrieve rooms, timeslots, student groups, teachers, and lessons (e.g., via Excel) will be added, streamlining data entry processes and ensuring accuracy. Furthermore, a feature to export the generated timetable will be included. Current issues related to the implemented constraints and updating entities using Spring JPA will be addressed to enhance overall system performance and accuracy.

Fine-tuning the algorithm and adding a filtered move selection for faster construction of the initial solution after the Construction Heuristics phase will be undertaken to improve efficiency. A benchmarking feature will be added, allowing users to test various configurations and evaluate their impact on performance. Finally, the application will be extended to enable centralized timetable generation for schools, high schools, and other colleges, each with potentially different constraints, thereby expanding the system's applicability and utility across various educational institutions.

9. Conclusions

The present research delves into the subject of automated timetable creation in an academic institution and examines the background, rationale, aims, and anticipated ramifications of the investigation. The objective of this study is to enhance the scheduling of educational tasks through the creation of an automated framework that produces effective and conflict-free schedules while satisfying the preferences and limitations of both faculty and students.

The present study concerns the optimization of didactic activities, which draws inspiration from the widely recognized timetable problem. The optimization of timetabling poses a complex challenge that pertains to the category of scheduling problems. They manifest in diverse fields, including education, healthcare, transportation, sports, and entertainment. The aforementioned issues are recognized as NP-complete, necessitating the utilization of heuristic methodologies to attain efficient resolutions.

The implementation of an automated system for generating timetables offers a multitude of advantages to an academic institution. The implementation of automation technology leads to a reduction in manual labor, a decrease in the occurrence of errors, and an optimization of resource allocation, ultimately resulting in heightened levels of efficiency and productivity. The focus of this study on tackling the obstacles posed by a range of distinct requirements, limitations, and preferences makes a valuable contribution to the domains of computer science, operations research, and artificial intelligence.

To accomplish the aims of this study, a proposed model has been put forth to generate an automatic timetable system. The implemented model integrates a web-based platform that has been constructed utilizing Angular to facilitate the frontend, Java in conjunction with Spring Boot to support the backend, and the Timefold Solver API for the timetable generation algorithm. This study has considered the architectural design, methodologies, algorithms, and appropriateness of a database.

In addition, a research methodology has been delineated to provide direction for this study. This methodology encompasses problem analysis, literature review, system design, implementation, data collection and integration, performance evaluation, and iterative

refinement. The utilization of a methodology guarantees a methodical and thorough strategy to effectively tackle the research objectives and challenges.

Regarding performance analysis measures, the evaluation metrics have been deliberated upon to appraise the excellence of the produced timetables, computational efficiency, user contentment, comparison with manual timetabling, empirical experimentation, and data analysis methodologies. These metrics offer valuable perspectives on the efficacy, productivity, and user-friendliness of the system, facilitating enhancements and refinements.

The results obtained from the experiments demonstrate the effectiveness and limitations of the algorithm on different hardware configurations. On the MacBook Pro M1 (2021), the algorithm achieved the best score of −284 init/0 hard/−2102 medium/7629 soft after eight hours and 0 hard/−2183 medium/8121 soft after twelve hours, indicating improvements in soft constraint scores but increased medium constraint violations over extended runtime. On the MacBook Air M1 (2020), shorter runtimes (5, 8, or 11 h) resulted in higher unassigned counts and lower soft constraint scores, necessitating at least 16 h to achieve a feasible solution. An alternative configuration that first assigned timeslots led to significant hard and medium constraint violations, suggesting a less effective approach.

In conclusion, this study on the implementation of automated approaches for creating timetables in a higher education institution demonstrates promise as an effective solution for the complex task of planning educational activities. By using an automated system and following a systematic study technique, it is feasible to improve resource allocation, minimize conflicts, and enhance overall efficiency. The proposed model, along with the performance analysis measures, provides a strong foundation for future research, implementation, and improvement in the automated scheduling system. Through continuous investigation and iterative improvements, it is feasible to achieve the desired outcome of improving the scheduling process and enhancing the educational experience for both teachers and students in academic institutions.

Author Contributions: Conceptualization, F.P.D. and C.T.; formal analysis, F.P.D. and C.T.; investigation, F.P.D. and C.T.; methodology, F.P.D. and C.T.; software, F.P.D.; supervision, C.T.; validation, F.P.D. and C.T.; writing—original draft, F.P.D.; writing—review and editing, C.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are available in a publicly accessible repository. The data presented in this study are openly available on GitHub at https://github.com/PatrickDiallo23/Automatic-Timetable-Generation-System (accessed on 1 September 2024).

Conflicts of Interest: Authors Francis Patrick Diallo and Cătălin Tudose were employed by the Luxoft Romania. The company was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

References

1. Knauer, B.A. Solution of a Timetable Problem. *Comput. Oper. Res.* **1974**, *1*, 363–375. [CrossRef]
2. Petrovic, S.; Burke, E. University timetabling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*; CRC Press: Boca Raton, FL, USA, 2004; pp. 1–23.
3. Pinedo, M.L. Planning and scheduling in manufacturing and services. In *Springer Series in Operations Research*; Springer: New York, NY, USA, 2005; pp. 3–8.
4. Fang, H.L. Genetic Algorithms in Timetabling Scheduling. Ph.D. Thesis, University of Edinburgh, Edinburgh, UK, 1994.
5. D'souza, D.; D'sa, O.; Pillai, P.; Shaikh, P. Multi-Constraint Satisfaction and Solution Optimization Using Genetic Algorithm for Solving Timetable Generation Problem. In Proceedings of the International Conference on Recent Advances in Computational Techniques (IC-RACT), Mumbai, India, 26–27 June 2020.
6. Teoh, C.-K.; Abdullah, M.Y.C.; Haron, H. Effect of Pre-Processors on Solution Quality of University Course Timetabling Problem. In Proceedings of the 2015 IEEE Student Conference on Research and Development (SCOReD), Kuala Lumpur, Malaysia, 13–14 December 2015; pp. 472–477.

7. Wungguli, D.; Nurwan, N. Application of Integer Linear Programming Model in Automatic Lectures Scheduling Optimization. *BAREKENG J. Math. App.* **2020**, *14*, 413–424. [CrossRef]
8. Schrijver, A. *Theory of Linear and Integer Programming*; Wiley: Hoboken, NJ, USA, 1998.
9. Feng, X.; Lee, Y.; Moon, I. An Integer Program and a Hybrid Genetic Algorithm for the University Timetabling Problem. *Optim. Methods Softw.* **2016**, *32*, 625–649. [CrossRef]
10. Schaerf, A. A Survey of Automated Timetabling. *Artif. Intell. Rev.* **1999**, *13*, 87–127. [CrossRef]
11. Bertsimas, D.; Tsitsiklis, J. Simulated Annealing. *Stat. Sci.* **1993**, *8*, 10–15. [CrossRef]
12. Pirim, H.; Eksioglu, B.; Bayraktar, E. Tabu Search: A Comparative Study. In *Tabu Search*; InTech Open: London, UK, 2008. [CrossRef]
13. Socha, K.; Sampels, M.; Manfrin, M. Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In *Applications of Evolutionary Computing, Proceedings of the EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK, 14–16 April 2003*; Cagnoni, S., Johnson, C.G., Romero Cardalda, J.J., Marchiori, E., Corne, D.W., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G.R., et al., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2611.
14. Abuhamdah, A.; Ayob, M. MPCA-ARDA for Solving Course Timetabling Problems. In Proceedings of the 2011 3rd Conference on Data Mining and Optimization (DMO), Putrajaya, Malaysia, 28–29 June 2011; pp. 171–177.
15. Abuhamdah, A.; Ayob, M. Multi-neighbourhood Particle Collision Algorithm for Solving Course Timetabling Problems. In Proceedings of the 2009 2nd Conference on Data Mining and Optimization, Selangor, Malaysia, 27–28 October 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 21–27.
16. Abuhamdah, A.; Ayob, M. Adaptive Randomized Descent Algorithm for Solving Course Timetabling Problems. *Int. J. Phys. Sci.* **2010**, *5*, 2516–2522.
17. Wankhede, S. Automatic College Timetable Generation. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2019**, *5*, 521–527.
18. Ong, B.C. Web-based Scheduling Software for a University. Ph.D. Thesis, Universiti Tunku Abdul Rahman, Kampar, Malaysia, 2020.
19. Atef Yekta, H.; Day, R. Optimization-Based Mechanisms for the Course Allocation Problem. *INFORMS J. Comput.* **2020**, *32*, 641–660. [CrossRef]
20. Burgess, R. Automation for CSUN Computer Science Department Course Scheduling. Ph.D. Thesis, California State University, Northridge, CA, USA, 2017.
21. OptaPlanner Official Website. Available online: https://www.optaplanner.org/ (accessed on 1 September 2024).
22. Timefold Official Website. Available online: https://timefold.ai/ (accessed on 1 September 2024).
23. Fiechter, A. University Timetable Scheduling. Bachelor's Thesis, Università della Svizzera Italiana, Lugano, Switzerland, 2018.
24. Grant, A. The Basics of AngularJS. In *Beginning AngularJS*; Apress: Berkeley, CA, USA, 2014; pp. 35–45.
25. Kunz, G. *Mastering Angular Components: Build Component-Based User Interfaces Using Angular*; Packt Publishing Ltd.: Mumbai, India, 2018.
26. Wilken, J. *Angular in Action*; Simon and Schuster: New York, NY, USA, 2018.
27. Arnold, K.; Gosling, J.; Holmes, D. *The Java Programming Language*, 4th ed.; Addison-Wesley Professional: Glenview, IL, USA, 2005.
28. Spring Boot Documentation. Available online: https://docs.spring.io/spring-boot/index.html (accessed on 1 September 2024).
29. Walls, C. *Spring in Action*; Simon and Schuster: New York, NY, USA, 2022.
30. Carnell, J.; Sánchez, I.H. *Spring Microservices in Action*; Simon and Schuster: New York, NY, USA, 2021.
31. Fielding, R.T. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Thesis, University of California, Irvine, CA, USA, 2000.
32. Spilcă, L. *Spring Security in Action*, 2nd ed.; Manning: New York, NY, USA, 2024.
33. Tudose, C. *JUnit in Action*; Manning: New York, NY, USA, 2020.
34. Erder, M.; Pureur, P.; Woods, E. *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*; Addison-Wesley Professional: Boston, MA, USA, 2021.
35. Tudose, C. *Java Persistence with Spring Data and Hibernate*; Manning: New York, NY, USA, 2023.
36. Bonteanu, A.M.; Tudose, C. Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java Programs Using JPA, Hibernate, Spring Data JPA. *Appl. Sci.* **2024**, *14*, 2743. [CrossRef]
37. Mustață, A.I.; Tudoruț, T.R.; Morar, A. Route Optimizations Using Clustering and Genetic Algorithms. *U.P.B. Sci. Bull.* **2020**, *82*, 3–14.
38. Zurn, H.H.; Quintana, V.H. Generator Maintenance Scheduling via Successive Approximations Dynamic-Programming. *IEEE Trans. Power Appar. Syst.* **1975**, *94*, 665–671. [CrossRef]
39. Mandel, R.A. Contribution to Solution of Job-Shop Scheduling Problems. *Ekon. Mat. Obz.* **1975**, *11*, 22–27.
40. Timefold Construction Heuristics Documentation. Available online: https://docs.timefold.ai/timefold-solver/latest/optimization-algorithms/construction-heuristics#constructionHeuristicsOverview (accessed on 1 September 2024).
41. Timefold Exhaustive Search Documentation. Available online: https://docs.timefold.ai/timefold-solver/latest/optimization-algorithms/overview#architectureOverview (accessed on 1 September 2024).
42. Timefold Optimization Algorithms Overview. Available online: https://docs.timefold.ai/timefold-solver/latest/optimization-algorithms/overview (accessed on 1 September 2024).

43. Timefold Constraints. Available online: https://docs.timefold.ai/timefold-solver/latest/introduction/introduction#aPlanningProblemHasConstraints (accessed on 1 September 2024).
44. Timefold Local Search Documentation. Available online: https://docs.timefold.ai/timefold-solver/latest/optimization-algorithms/local-search (accessed on 1 September 2024).
45. Ohashi, T.; Aghbari, Z.; Makinouchi, A. Hill-Climbing Algorithm for Efficient Color-Based Image Segmentation. In Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition, and Applications, Innsbruck, Austria, 30 June–2 July 2003; pp. 17–22.
46. Gendreau, M.; Potvin, J.Y. *Tabu Search. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 165–186.
47. Aarts, E.; Korst, J.; Michiels, W. *Simulated Annealing, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 187–210.
48. Brailsford, S.C.; Potts, C.N.; Smith, B.M. Constraint satisfaction problems: Algorithms and applications. *Eur. J. Oper. Res.* **1999**, *119*, 557–581.
49. Kalshetti, U.; Nahar, D.; Deshpande, K.; Gawas, S.; Sudeep, S. Dynamic Timetable Generation Using Constraint Satisfaction Algorithm. In Proceedings of the Second International Conference on Computer and Communication Technologies, Hyderabad, India, 24–26 July 2015; Volume 379, pp. 761–771.
50. Ferrari, L.; Pirozzi, E. *Learn PostgreSQL: Use, Manage and Build Secure and Scalable Databases with PostgreSQL 16*, 2nd ed.; Packt Publishing: Mumbai, India, 2023.
51. Juba, S.; Vannahme, A.; Volkov, A. *Learning PostgreSQL*; Packt Publishing Ltd.: Mumbai, India, 2015.
52. Wu, S.; Kemme, B. Postgres-R (SI): Combining replica control with concurrency control based on snapshot isolation. In Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, 5–8 April 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 422–433.
53. Shukla, S.; George, J.P.; Tiwari, K.; Kureethara, J.V. Data Security. In *Data Ethics and Challenges*; Springer: Singapore, 2022; pp. 41–59.
54. Park, J.S.; Sandhu, R.; Ahn, G.J. Role-Based Access Control on the Web. *ACM Trans. Inf. Syst. Secur. TISSEC* **2001**, *4*, 37–71. [CrossRef]
55. *RFC 7519*; JSON Web Token (JWT). IETF: Washington, DC, USA, 2015.
56. Anghel, I.I.; Calin, R.S.; Nedelea, M.L.; Stanica, I.C.; Tudose, C.; Boiangiu, C.A. Software development methodologies: A comparative analysis. *UPB Sci. Bull.* **2022**, *83*, 45–58.
57. Timefold Annotations. Available online: https://docs.timefold.ai/timefold-solver/latest/using-timefold-solver/modeling-planning-problems (accessed on 1 September 2024).
58. Forgy, C.L. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In *Readings in Artificial Intelligence and Databases*; Morgan Kaufmann: Burlington, MA, USA, 1989; pp. 547–559.
59. Ghomi, E.J.; Rahmani, A.M.; Qader, N.N. Load-Balancing Algorithms in Cloud Computing: A Survey. *J. Netw. Comput. Appl.* **2017**, *88*, 50–71. [CrossRef]