An Automated Solution to Exam Scheduling at Østfold University College

Ole M. Hansen, Radin Morik, Shvan Nasser, Stine Vågnes, and Monica Holone

Østfold University College, Halden, Norway
{omhanse, radinm, shvann, stineva, monica.kristiansen}@hiof.no

Abstract. Exam scheduling at universities is often a complex and time-consuming task. At Østfold University College (HIOF), this task is currently performed manually. An exam coordinator must handle constraints, such as ensuring at least 3 days between mandatory course exams and adhering to pre-assigned dates from the Exam Office. This manual approach is prone to human error and can result in scheduling conflicts. To address this exam scheduling problem, we developed a user-friendly application paired with a greedy incremental algorithm. The proposed algorithm takes last year's schedule as input and iteratively moves exams that violate constraints by 1 day per step. The algorithm is easy to implement and ensures that key constraints are met. On a computer with average specifications, our application produces a conflict-free schedule in seven milliseconds. Any remaining conflicts can be manually fixed through the spreadsheet. Furthermore, the algorithm is paired with a Java Swing framework to develop a user-friendly graphical user interface (GUI) for the application. In its current state, HIOF suggests the application will significantly reduce manual work, lower the risk of human error, and bring HIOF to a near-fully automated exam scheduling process.

Keywords: Exam scheduling · Timetabling · Local search algorithm · User-centered design

1 Introduction

Exams are the primary method universities worldwide use to evaluate students' academic performance [1,2]. These examination periods usually span three to six weeks and can lead to tightly packed schedules, which exam coordinators must navigate with care. After assigning a temporary schedule to each course, conflicts frequently occur, creating a scheduling problem that can be difficult to solve manually. In large departments such as Information Technology (IT), hundreds of students take numerous courses with different exams for each, with changing requirements and constraints each semester [3]. Exam coordinators must balance student availability and specific needs while adhering to the instructions from the Exam Office. This manual process can often result in exam schedules that still have room for further optimizations to reduce the number of examination conflicts [1].

The exam scheduling problem can negatively affect a student's academic performance. Small time windows between exams and possible conflicts are likely due to tight scheduling. Pope and Fillmore [4] show in their research that student performance is associated with the time interval between exams. Their estimates suggest that a student taking two exams ten days apart is 6-8% more likely to pass both than only having one day between the exams. This shows that if the exam schedule is optimized with as few conflicts as possible, it can result in better academic performance for students.

The exam coordinator at Østfold University College (HIOF) currently schedules all exams manually, a process that is both time-consuming and open to improvement. To address this problem, we developed an automated exam scheduling application that can be used by individuals without an IT background and is tailored to the Exam Office's specific exam-planning instructions. These include having a minimum of three days between obligatory course exams within a study program, avoiding weekend exams, and accommodating specific preassigned exam dates. Although HIOF aims to reduce the amount of manual work involved in scheduling, they still desire a degree of manual control over the final schedule. The proposed solution developed in this paper helps automate HIOF's current exam scheduling by utilizing a greedy incremental algorithm. The algorithm uses an exam plan from the previous semester as input and adjusts it until all constraints are met. At the request of HIOF, the algorithm was designed to generate a complete schedule within one minute. To make the system accessible to the exam coordinator, we developed an intuitive and user-friendly application alongside the algorithm.

The key contributions of this paper are as follows.

1. A smart exam planning algorithm was developed that automates the HIOF exam planning process. Compared to more complex approaches, such as genetic algorithms and graph coloring, our greedy incremental approach is easy to implement for small to medium-sized datasets.

2. A user-friendly application was developed alongside the algorithm to improve usability and productivity for exam planners.

3. An application that significantly reduces the manual work from about two days to a few minutes.

Our proposed solution shows promise and is currently being evaluated by HIOF for future exam planning. Initial tests using past semester schedules show that our algorithm produces valid and accurate plans with a runtime of seven milliseconds.

2 Related Work

This section reviews the state-of-the-art in exam scheduling at HIOF and the findings from previous research. We highlight the solutions proposed by other researchers, with particular attention to their applicability to the problem statement of this paper. By identifying gaps in the existing literature, we clarify how our work contributes to this field.

The current exam scheduling process at HIOF is performed manually by the exam coordinator. Rather than automated tools, they rely on spreadsheets and slides for adjustments. The Exam Office provides the exam coordinator with a spreadsheet of suggested exam dates, which the coordinator then transfers to multiple presentation slides to visualize and rearrange. This approach allows schedules to exist in multiple places, which can lead to human errors when updates in one document are not reflected in others. Through interviews with the exam coordinator, we learned that this manual process is time-consuming and frustrating, underscoring the need for a simple, automated alternative.

Researchers have attempted to solve the exam timetable problem using primarily genetic algorithms [1, 3, 5, 6] and graph coloring algorithms [2, 7]. Xiaofan et al. [1] propose a solution that uses an improved genetic algorithm to sort all exams and dates. Their results show that their proposed algorithm demonstrated superior problem-solving performance compared to traditional genetic algorithms and manual exam scheduling. However, the algorithm needed to allow for some conflicts. To reduce run-time and scheduling conflicts, Jáhn-Erdös and Köváti [5] investigate the usage of generic algorithms. They adjust factors such as mutation rates and population size. In contrast to our main focus on student-centered constraints, such as ensuring a minimum of three days between exams, their approach is less appropriate for our paper as it prioritizes examiner availability and room capacity. Similarly, Muklason et al. [7] optimize exam scheduling at universities and minimize penalties related to generic scheduling conflicts by using graph coloring and a hyper-heuristic method. Although effective in general situations, their solution is less applicable in our case since it ignores institutional requirements specific to HIOF. Elen [6] proposes a genetic algorithm-based solution to the timetable problem on a much larger scale, accounting for classrooms and exam times. Their solution allows for more flexible constraints on overlapping exam dates and on students retaking failed courses. However, run times can be inconsistent based on the level of constraints set by the user. Furthermore, mathematical optimization techniques such as Group Role Assignment with Constraints (GRA+) have been suggested to guarantee schedules free of conflicts [3].

Despite many existing works on exam planning, most are not directly relevant to the problem statement addressed in this paper. Their proposed solutions are often too complex and designed for larger-scale datasets beyond the scope of this paper. Furthermore, none of the previous solutions focus on developing a user-friendly application alongside the algorithm, which is a focus of this paper.

Although these methods handle general scheduling constraints well, some overlook specific institutional requirements. These constraints include a three-day gap between exams and avoiding multiple exams on the same day. Such constraints are crucial to our goals for the exam scheduling algorithm. This gap in existing research is where our work contributes by offering a user-friendly solution that is suited to institutional needs that are often overlooked in existing approaches.

3 Proposed Solution

To solve the exam scheduling problem at HIOF, a user-friendly application was developed, along with a greedy, incremental algorithm to schedule exams. The proposed solution was developed following an Agile working process and a continuous development cycle through meetings with the university college [8]. This flexible, user-centered approach ensured that expectations and constraints specific to the exam-planning process at HIOF were met.

3.1 Prototyping and Design Iterations

Open interviews with the exam coordinator at HIOF were conducted to evaluate the application's performance. The interviews helped uncover new ideas and contextual insights that could have been difficult to capture through structured interviews [9]. In addition to interviews, usability tests were performed to evaluate the user-friendliness of the application [10].

Low-fidelity wireframes were made in Figma[1] to obtain a visual understanding of how the application should look before implementation [12]. The GUI was inspired by the state-of-the-art to keep the application familiar to the exam coordinator. The wireframes were based on HIOF's needs, which were identified during the interviews. The resulting design includes two layouts for viewing exams, as well as filtering options for exam types and courses. The Figma sketch was presented to the exam planners for feedback and suggestions. After the low-fidelity prototype was finalized, a high-fidelity prototype was developed in Java[2] for further usability testing.

3.2 Constraints

A set of critical scheduling constraints was discovered through interviews with the exam coordinator. The algorithm must comply with these constraints to generate a valid exam schedule. An examination conflict is defined as any breach of constraints listed below, such as multiple exams of the same type (written/project submission) on the same day. The concept of 'exam' in this paper

[1] Figma is a web-based user interface design and prototyping tool that enables real-time collaboration. See [11] for details.

[2] Java is a general-purpose, object-oriented programming language. See [13] for details.

includes two main types: written exams and project submissions. The algorithm considers no other exam categories. The instructions given by the Exam Office primarily refer to the six constraints listed. These constraints only apply within a single faculty and do not include classroom allocation, which was outside the scope of this paper. The constraints are presented below:

(1) Exams cannot be scheduled on public holidays or weekends.

(2) Multiple written exams can not be scheduled on the same day.

(3) Multiple project submissions can not be scheduled on the same day.

(4) Exams that are mandatory within each program need a minimum of three days between each other.

(5) Some exams may be locked to specific dates and should not be moved.

(6) All exams must be scheduled within the official examination period, which spans three to five weeks.

The constraints listed above directly shaped how the algorithm evaluates exam schedules and conflicts.

3.3 Proposed Application

The application was developed with Java using the Swing[3] framework to provide a user-friendly graphical user interface (GUI) based on the findings from the design iterations. The application provides both drag-and-drop and traditional file-chooser for loading a spreadsheet of exam data, including course names, course codes, exam dates, and other necessary information. Users can view a single schedule or compare two schedules side by side. Before rendering the exam table, the algorithm checks for constraint breaches, flagging conflicting exams. Flagged exams are shown in red text in the GUI. To resolve the conflicts, the algorithm can be run by pressing the "reschedule" button.

3.4 Data Preprocessing

To prepare the application input, the previous year's exam data from HIOF had to be preprocessed. This involved consistent date formatting and the exclusion of irrelevant columns. The Exam Office will need to maintain this format to ensure the application correctly parses the spreadsheet. The application takes the preprocessed spreadsheet as input, where each row represents an individual exam. An import module was created to parse each spreadsheet row into an Exam object. Each exam contains the course ID and name, the IDs of the programs for which the exam is mandatory, the examination type, a suggested exam date, and a value indicating whether the exam date is locked.

[3] Swing is Java's standard GUI toolkit, part of the Java Foundation Classes. See [14] for details.

Upon import, the spreadsheet is restructured into a format acceptable to the algorithm. All the exam dates are stored as an array of DateNode objects. Each DateNode includes variables for whether it is a weekend or a public holiday, as well as a list of all exams scheduled for that day. Exams with missing or incorrectly formatted dates will be placed at the end of the exam period. Other missing values in the exams will cause the exam to be excluded and give an alert message on import. Preprocessing was essential to ensure the algorithm's efficiency and correct output.

3.5 Proposed Greedy Incremental Scheduling Algorithm

The proposed algorithm, called the Greedy Incremental Scheduling (GIS) algorithm, is shown in Algorithm 1. The algorithm utilizes a suggested exam plan, typically the plan of the previous year, and produces a new plan for the current year. To produce a plan that satisfies the constraints, the algorithm incrementally improves it in multiple steps. In each step, the exams with constraint breaches are moved one day forward or backward, performing the local optimal choice. To account for dates being shifted one day forward each year, exams are preferred to be moved back to the previous date. Only moving exams one day at a time helps ensure the new exam plan maintains a similar structure to the original exam plan. This provides control over the output by manually adjusting the exam dates in the input spreadsheet. For exams that are added or removed from year to year, the input spreadsheet can be edited to represent these changes.

The GIS algorithm performs steps until there are no remaining moves, as shown in line 2. At each step, the algorithm iterates through the array of dates and the list of exams on each date. For each exam, the constraints are checked. To verify constraint (1), the algorithm checks if the date is marked as a weekend or a public holiday on line 6. To verify constraints (2) and (3), the algorithm checks if the exam is a written exam or a project submission on line 12. If so, the algorithm checks if there are any other exams of the same type on the same date, as shown in lines 13 and 14. To verify constraint (4), the algorithm checks if the exam is mandatory within any program. In that case, it checks for any other mandatory exams within three days in the same program. If any of the constraints are breached, the appropriate exam is moved to the previous or next date, provided it is not locked. This ensures that constraints (5) and (6) are followed.

Algorithm 1 Proposed Greedy Incremental Scheduling (GIS) Algorithm

```
1: Input: Exam schedule for the previous year and constraints
2: while moves > 0 do
3:    Set moves = 0
4:    for date = 1 to 60 do
5:       for exam = 1 to date.exams do
6:          if date is weekend or holiday then
7:             Move exam to previous date or next date
8:             Increase moves
9:          end if
10:         if exam is written or project then
11:            for other_exam = exam + 1 to date.exams do
12:               if other_exam is written or project then
13:                  if other_exam or exam is not locked then
14:                     Move other_exam or exam to previous date or next date
15:                     Increase moves
16:                  end if
17:               end if
18:            end for
19:         end if
20:         for program = 1 to exam.programs do
21:            for other_date = date to date + 3 do
22:               for other_exam = 1 to other_date.exams do
23:                  for other_program = 1 to other_exam.programs do
24:                     if program is other_program then
25:                        if exam or other_exam is not locked then
26:                           Move exam or other_exam to previous date to next date
27:                           Increase moves
28:                        end if
29:                     end if
30:                  end for
31:               end for
32:            end for
33:         end for
34:      end for
35:   end for
36: end while
37: Output: Exam schedule without conflicts
```

4 Performance Evaluation

This chapter evaluates the application's usability and the algorithm's runtime performance and complexity. The application was evaluated running on standard computers with modest specifications (e.g. 8 GB RAM and 4 core CPU 4Ghz). The application runs on OpenJDK 23[4] and was developed in the Java programming language using IntelliJ[5] as the primary IDE. Re-implementing the algorithm in lower-level languages such as C could reduce runtime. However, the added complexity would increase the difficulty of implementing the algorithm.

4.1 Evaluation of Algorithm

Picture of the application before running the algorithm

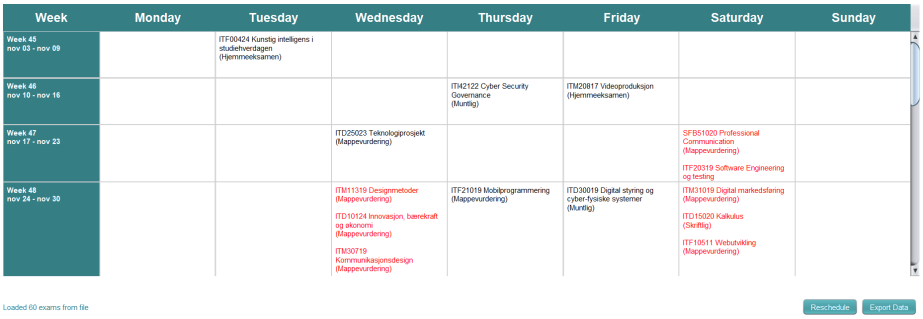

Fig. 1: Application Before

To ensure compliance with the previously defined constraints, the algorithm applies a set of actions. The following list details how each constraint is handled within the algorithm:

(1) The algorithm moves exams on public holidays or weekends to adjacent dates.

(2) The algorithm moves excess written exams on a single day to adjacent dates.

(3) The algorithm moves excess project submissions on a single day to adjacent dates.

(4) The algorithm moves exams that are mandatory within the same program, and within three days of each other, further apart.

(5) Exams can be marked as locked in the input and will then not be moved by the algorithm.

(6) Exams cannot be moved outside of the exam period by the algorithm.

[4] OpenJDK is an open-source implementation of the Java Platform, Standard Edition. See [15] for details.

[5] IntelliJ IDEA is an integrated development environment (IDE) developed by JetBrains. See [16] for details.

Picture of the application after running the algorithm

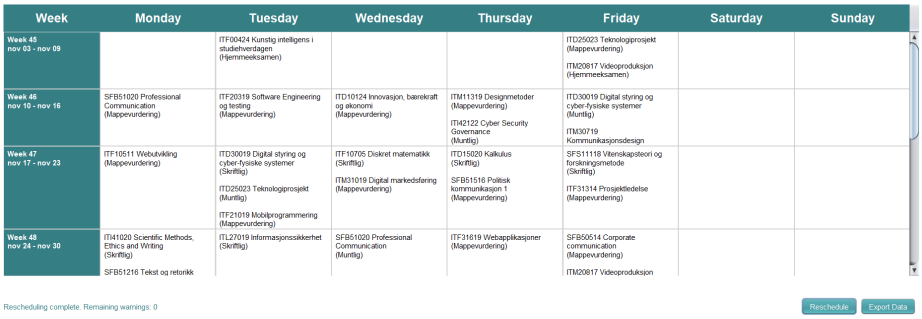

Fig. 2: Application After

The runtime of the algorithm on the provided dataset is seven milliseconds, which is well within HIOF's requirement of one minute. The runtime complexity of the algorithm is $\mathcal{O}(n^2)$. As the number of exams remains relatively constant each year, it is not expected that the dataset size will increase significantly. Therefore, the algorithm's runtime and complexity are sufficient to address the exam scheduling problem at HIOF.

As shown in Figures 1 and 2, the algorithm resolves all constraint breaches, marked with red text, in the provided dataset. Project submissions can be seen moved to separate days, and all exams are moved away from the weekends. Any remaining constraint violations that could occur with other input data can be fixed manually by editing the output spreadsheet. The application highlights these issues, enabling the exam coordinator to get an overview of conflicts quickly. The university college accepts the resulting output.

4.2 Usability of the Application

We conducted a usability test with the exam coordinator to evaluate the current GUI and functionalities of our application. They appreciated the application's clear layout, runtime performance, functionality, and use of HIOF's color palette. The user-friendly GUI made it easy to import spreadsheets via drag-and-drop functionality or the traditional file menu in the menu bar. Conflict highlighting made it easy to get a quick overview of exam collisions and constraint breaches. Filtering with the search bar and drop-down menu allows users to intuitively isolate specific study programs or exam types. Furthermore, the application keeps all information in one place, unlike the current manual exam scheduling process, which presents exam dates across multiple slides. The exam coordinator concluded that, as a scheduling tool, the application can be used alongside current manual work to make exam planning more efficient and less prone to human errors.

The exam coordinator suggested several improvements that should be implemented before the application can fully replace the manual exam scheduling process. These include: added constraints for exams spanning multiple days,

manual exam adjustments, and multi-keyword search. More GUI interactivity, such as marking public holidays, locking exams to specific days, and drag-and-drop exam rearrangement, would reduce reliance on spreadsheet editing. Improving the exam comparison view and the interface's responsiveness across screen sizes would further enhance usability. Implementing these refinements will ensure that the application not only accelerates exam scheduling but also meets HIOF's usability requirements.

5 Conclusion

In this paper, we addressed the exam scheduling problem at HIOF by developing an automated solution. We introduced an efficient, easy-to-implement greedy incremental algorithm that takes last year's schedule as input and iteratively adjusts conflicting exams, one day at a time. Paired with a Java Swing–based GUI, our application generates conflict-free schedules in seven milliseconds on a standard computer, highlighting any remaining constraint violations. Through Agile co-design with HIOF's exam coordinator, we ensured that the application is user-friendly and does not require users to have an IT background. Our proposed solution reduces work time from about two days to a few minutes, including seven seconds for the computer to run our algorithm and a few minutes for the coordinator to check and finalize the schedule. It also lowers the risk of human errors and moves HIOF closer to a near-fully automated exam scheduling process. Our evaluation demonstrates that the algorithm meets all the current constraints. The usability tests suggest that the application's current functionalities significantly simplify the exam coordinator's workflow. With future improvements, our application can further automate the remaining manual work. HIOF acknowledges that our current application can be a helpful tool for significantly improving the exam scheduling process.

Acknowledgments

We want to thank our supervisor, Thi Thuy Nga Dinh, whose guidance and expertise were invaluable throughout this paper.

References

1. L. Xiaofan, L. Xiaolong, and X. Jiaqian, "The exam timetable problem with examination conflicts," in *Proceedings of the 2023 6th International Conference on E-Business, Information Management and Computer Science*. Hong Kong, Hong Kong: ACM, dec 2023, pp. 128–133.
2. O. S. Egwuche, "Examination timetabling with graph coloring for emerging institutions," in *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*. Ayobo, Ipaja, Lagos, Nigeria: IEEE, mar 2020, pp. 1–6.
3. H. Zhu, Z. Yu, and Y. Gningue, "Solving the exam scheduling problem with gra+," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Toronto, ON, Canada: IEEE, oct 2020, pp. 1485–1490.
4. D. G. Pope and I. Fillmore, "The impact of time between cognitive tasks on performance: Evidence from advanced placement exams," *Economics of Education Review*, vol. 48, pp. 30–40, oct 2015.
5. S. Jáhn-Erdös and B. Kövári, "Exploring the potential of a genetic algorithm on a real-world complex scheduling problem," in *2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. Toronto, ON, Canada: IEEE, nov 2022, pp. 113–117.
6. A. Elen, "A complete solution to exam scheduling problem: A case study," *JSR-A*, no. 049, pp. 12–34, jun 2022.
7. A. Muklason, E. J. Pratama, and I. G. A. Premananda, "Generic university examination timetabling system with steepest-ascent hill climbing hyper-heuristic algorithm," *Procedia Computer Science*, vol. 234, pp. 584–591, 2024.
8. OpenText. (2025) What is agile development? Accessed: 2025-05-19. [Online]. Available: https://www.opentext.com/what-is/agile-development
9. T. George. (2022) Unstructured interview | definition, guide & examples. Accessed: 2025-05-03. [Online]. Available: https://www.scribbr.com/methodology/unstructured-interview/
10. S. Krug, *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*, 3rd ed. Berkeley, Calif.: New Riders, 2014.
11. Figma, "Figma," 2025, accessed: 2025-05-03. [Online]. Available: https://www.figma.com
12. T. Bratteteig, *Design for, med og av brukere: å inkludere brukere i design av informasjonssystemer*. Oslo: Universitetsforlaget, 2021.
13. Oracle, "Java," 2025, accessed: 2025-05-03. [Online]. Available: https://www.oracle.com/java/
14. ——, "Java swing documentation," 2025, accessed: 2025-05-03. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/swing/
15. OpenJDK, "The openjdk project," 2025, accessed: 2025-05-03. [Online]. Available: https://openjdk.org
16. JetBrains, "Intellij idea," 2025, accessed: 2025-05-03. [Online]. Available: https://www.jetbrains.com/idea/