OST
Eastern Switzerland
University of Applied Sciences

Fabian Germann & Raphael Jenni

Automation of the OST-RJ Examination Scheduling

Bachelor's Thesis

OST – Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Supervision

Prof. Dr. Farhad Mehta

June 2021

Abstract

As stated in our prior semester project ”Automation of the OST-RJ Examination Scheduling”, December 2020, exam scheduling is a known NP-complete problem. Finding the best solution for such a problem is nearly impossible for a human and takes forever for a computer. Computer-aided exam scheduling, or generally speaking problem-solving, takes advantage of trying many possible solutions in an automated way and combining it with algorithms that help optimize the solving process. Testing the quality of a solution is carried out based on several constraints and their assigned penalties and weights. Precisely tailored mathematical calculations and carefully chosen algorithms are required for providing solutions that meet the requirements in a reasonable amount of time.

The version developed for the bachlor's thesis expands the functionality of the exam scheduler not only to cover the mandatory constraints but also to optimize the solution based on availabilities, optimal distributions, and nice-to-have properties such as lunch breaks. Furthermore, considerable improvements have been made to the user interface, allowing the user to import data, control the solving process, visualize the exam schedule, gain insights into the resulting solution, and manually modify the schedule to work with the application hand in hand.

The software and its results are ready for a first pilot phase. Although some manual preprocessing of the exam schedule is still required, scheduling becomes much more comfortable, reliable, and of much higher quality. The process of examination scheduling is reduced from several weeks to an absolute minimum in the range of days or even hours. Our comparison of automatically generated examination schedules with manually created ones shows at least equal if not better results.

Keywords: Exam Scheduling, Problem Solving, Constraint Programming, OptaPlanner, NP-Completeness, Metaheuristics.

Executive Summary

As described in our prior semester project ”Automation of the OST-RJ Examination Scheduling”, December 2020, exam scheduling is known to be a challenging problem to solve. Manual scheduling of the exams is executed in an enormous Excel file and takes several weeks. The objective of the present bachelor project/thesis is to provide a fully functional version of an automated, computer-aided exam scheduling solution.

In the exam scheduler’s current version, all rules defined as mandatory for an acceptable solution by the customer and several rules for optimizing the resulting exam schedule from the students’ point of view are implemented. The rules are defined using the constraint solver OptaPlanner, an open-source library maintained by RedHat. The lists of exams and students are uploaded as files, validated, and then imported, pre-processed, scheduled, and finally exported as files or directly accessed via an API. A web application provides functionalities such as the possibility of visualizing the resulting schedule, manually setting specific exams, defining room availabilities, and getting insights into the quality of the resulting exam schedule.

The current solution is ready for a first pilot phase. Although some manual pre-processing of the exam schedule is still required, scheduling becomes much more comfortable, reliable, and of much higher quality. The process of examination scheduling is reduced from several weeks to an absolute minimum in the range of days or even hours. Our comparison of automatically generated examination schedules with manually created ones shows at least equal if not better results.

For further enhancements of the developed solution, the end-users have to conduct in-depth tests and document their findings. The constraint weights might need to be adjusted, and additional constraints or features in the web application to be added in order for the exam scheduler to be fully production-ready. With the continuation of the project, a fully automated scheduling process can be achieved and might even be expanded to other schools. It promises significant improvements in the quality of the schedules and the time and iterations it takes to create solid exam schedules.

Acknowledgment

Throughout our bachelor's thesis and the prior semester project, many people supported us in one way or the other.

First, we want to thank our supervisor, Professor Dr. Farhad Mehta, whose expertise in defining the project scope and helping us to find the right starting point was invaluable. Your feedback and in-depth questions motivated us to strive to achieve what we did and brought our work to a higher level.

We want to thank our client, Professor Dr. Stefan Martignoli, for working so closely and interactively with us. Your immediate feedback, openness to changes, and ideas to improve the software led to the tool's quality we have now.

Many thanks to Professor Dr. Andreas Müller, who provided us his server for running our tests, gave us the opportunity to present our project as part of his lecture, and proofread this thesis.

Many thanks to AnneMarie O'Neill and Christophor Jenni for proofreading our documentation and giving us valuable feedback regarding the language and form of this thesis.

Contents

Chapter 1

Introduction

Roughly 250 exams involving up to 1,500 students take place on the OST campus in Rapperswil-Jona each semester. Creating an examination schedule on this scale is not an easy task. Many spatial and temporal constraints have to be considered in order to find a solution that can be seen as a good and balanced examination schedule.

Until now, all exam timetables have been created manually by the examination planning team. Exam scheduling by hand is not only complicated and tedious but also takes a lot of time. Therefore, the examination planning team aims to automate this process with the help of a software solution. The first step regarding an exam scheduler software has already been taken. This bachelor thesis is a follow-up project of our semester project "Automation of the OST-RJ Examination Scheduling", in which we evaluated a constraint-programming-based approach as the most promising one to tackle the exam scheduling problem. We implemented the first version of the exam scheduler software, which uses the OptaPlanner as a constraint solver underneath. With the result of our semester project, we have shown that the software can generate examination schedules. Furthermore, we have demonstrated the potential to generate better schedules than a human could create by hand. However, the resulting problem model is not production-ready as it does not fully meet all requirements yet [1].

1.1 Objective

This thesis has two primary objectives: First, we want to develop the existing exam scheduler further so that the examination planning team can use it productively. To achieve this goal, we must improve the problem model by including the missing requirements and implement some additional features to turn the application into a helpful instrument that supports the examination planning team in their work. To make all features easily accessi-

ble for the users, we must provide a GUI. Finally, an interface between the exam scheduler and the university's computer system is needed, enabling the academic services department to use the generated examination schedules with their administration tools.

The second objective is the evaluation of different optimization algorithms used by the constraint solver (OptaPlanner) as well as the parameterization/weighting of our defined cost functions. We want to examine its impact on the schedule quality, solving time, and convergence behavior with the goal to choose the most efficient configuration that leads to satisfying results.

1.2 Structure of the Thesis

The thesis is organized as follows:

Chapter 2 – Problem Analysis gives an overview of the domain and describes the state and functionality of the exam scheduler developed during the previous semester project. It also lists all constraints that have to be taken into account for an optimal examination schedule. This list states which constraints the problem model already covers and which are left to be implemented in this project.

Chapter 3 – Research describes the development of the cost function for an optimal exam distribution. Moreover, it covers details regarding the refinement of the problem model for the solver, and the literature research that was conducted regarding the different optimization algorithms it can use.

Chapter 4 – Solution describes the key improvements of the exam scheduler, performance optimizations made, application and UI design decisions, the data handling, the applied quality assurances, and a detailed explanation of the score function.

Chapter 5 – Results presents and discusses the results of the bachelor's thesis. The score function is evaluated with the help of key performance indicators. It also provides a comparison of a generated exam schedule with one made by a human.

Chapter 6 – Conclusion summarizes the outcome of the bachelor's thesis. In addition, an outlook for further possible improvements and features regarding the developed software is given.

Chapter 2

Problem Analysis

2.1 Domain

The problem domain (Figure 2.1) is simple. Students take exams. For each exam, there is precisely one examiner in charge. This person is responsible for the exam in general but must not necessarily supervise the exam. Supervisors are not part of the problem domain because they are usually selected after the examination schedule has been created. The overall goal is to create an exam timetable that schedules all exams. The scheduling of a single exam is represented by a timetable entry. It assigns a date and one or multiple rooms for each exam.

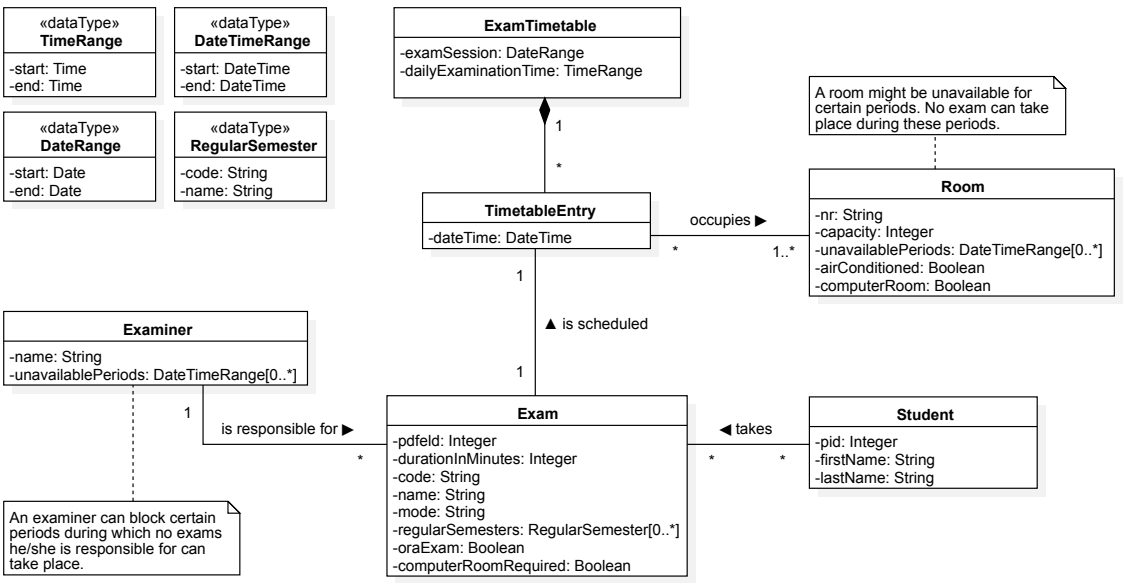


Figure 2.1: Domain Model

A room might not be available for certain periods as it is used for other purposes. Exams can only take place in available rooms. Similarly, an examiner in charge can block certain time periods during which the exam must not take place.

Each exam is assigned to one or more regular semesters. The exam usually takes place in these semesters. In other words, students usually write the exam in these semesters. Each degree program has its own regular semesters. For instance, the full-time degree program “Computer Science” has in total six regular semesters. Regular semesters are used in order for the exams to be optimally distributed. Details regarding its impact are discussed in section 5.3 [1].

2.2 Overview of the Existing Exam Scheduler Software

This section relates to the software developed in the semester project [1]. For readability reasons, we omit all further references to it.

Software architecture: The container diagram (Figure 2.2) shows the high-level software architecture. The core is an API application running Spring Boot, which provides the exam scheduler capabilities. An Angular single-page application allows the user to interact with the system. The single-page application, as well as the API documentation, is served by the Spring Boot container. A PostgreSQL instance manages and stores the data.

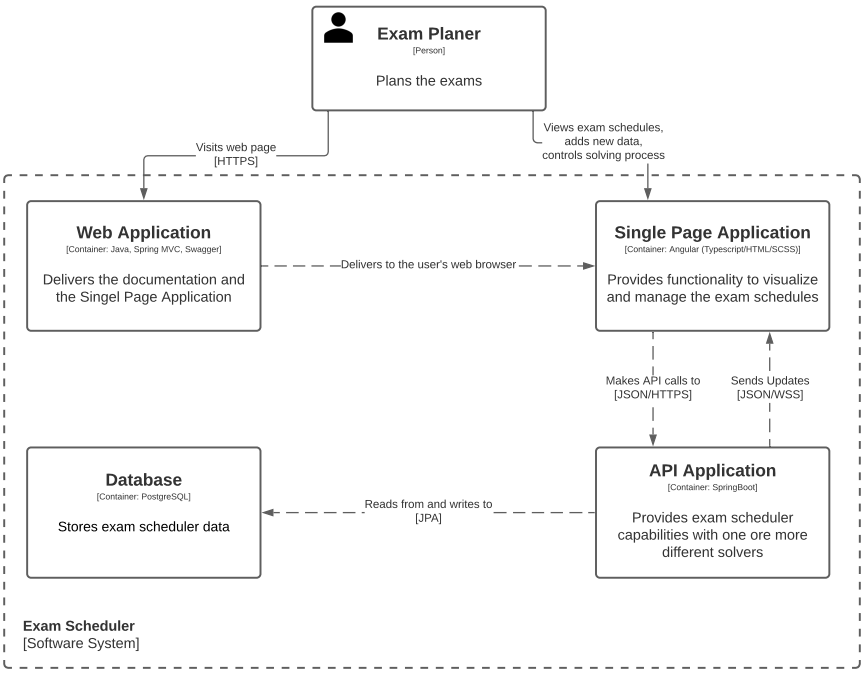


Figure 2.2: C4 – Container Diagram

User interface: The user interface developed in the semester project (Figure 2.3) is limited in functionality. During the semester project, we mainly used it to visualize generated examination schedules. Creating new exam timetables and managing the solving process is only available via the web API. Alternatively, there is the possibility to use the web-based OpenAPI documentation, which is not very user-friendly.

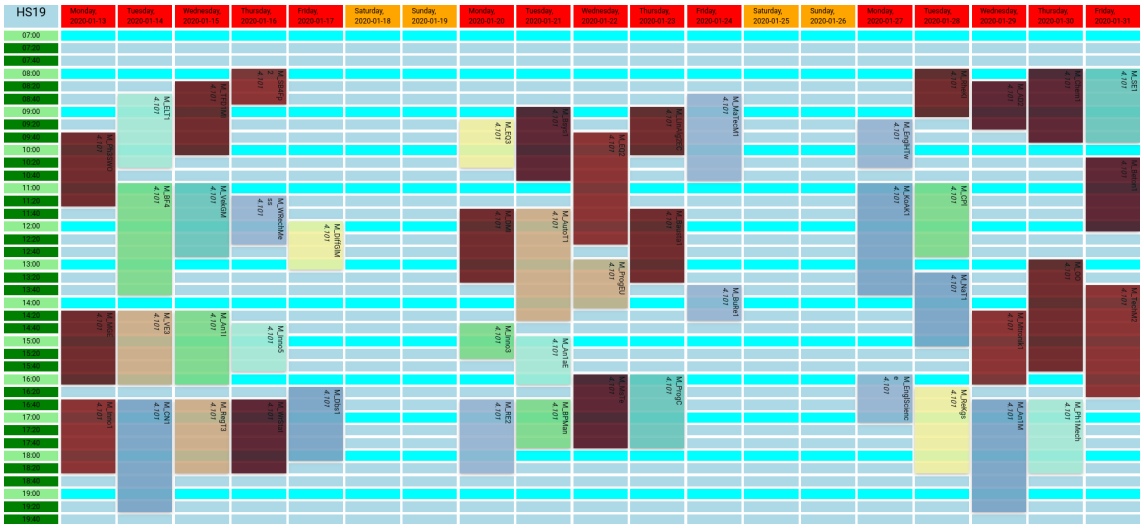


Figure 2.3: Old Exam Scheduler User Interface: All exams that take place in the school hall (room 4.101) are visualized in an exam timetable.

Constraint solver: The exam scheduler uses a constraint solver to search for the best possible examination schedule. In constraint programming, the problem is described in a declarative manner using variables and relationships between them. Then a constraint solver takes over the solution process [2]. In the case of the exam timetable problem, the task is to allocate exams in time and space (rooms) respecting various constraints and to satisfy a set of desirable objectives as well as possible.

The exam scheduler is explicitly designed to support multiple, different constraint solvers. However, replacing or adding a constraint solver requires a lot of work as each one has a different API and needs its specific problem model. In our semester project, we implemented the OptaPlanner as a constraint solver. The capabilities of the current problem model are described in section 2.3.

Data import and export: The input data is given as multiple Excel files. In the semester project, we implemented a data import API that validates the input data and converts it into the application data model. The generated examination schedules can be exported as a CSV or JSON file.

2.3 Constraints

At the very beginning, the examination planning team gave us a list of the constraints for an optimal examination schedule. Together we have refined them based on the results of our semester project. Below is an overview of all constraints. The constraints that we already had implemented in the semester project are marked with a checkmark (☑).

We distinguish two types of constraints. Hard constraints are constraints that cannot be broken. For example, a student cannot write two exams at the same time. If an exam timetable fulfills all hard constraint we call it a *feasible solution*. The second type are soft constraints. These constraint can be broken but should be fulfilled for an *optimal solution*.

Hard constraints:

- ☑ Students cannot have two exams at the same time.
- ☑ Students can only have two exams that take up to two hours or one exam that takes more than two hours on the same day.
- ☑ Students need a break of at least two hours between exams.
- ☑ Examiner cannot have two exams at the same time.
- ☑ There can only be one exam in a room at a time.
- ☑ The room needs to have the capacity for at least two students more than registered.
- ☑ There has to be a break of at least 20 minutes between two exams.
- ☐ In summer, exams after lunch may only take place in rooms with air conditioning.
- ☐ Computer-based exams can only take place in computer rooms.

Soft constraints (sorted descending in priority):

- ☐ The exams of a regular semester should be distributed as evenly as possible over the examination session.
- ☑ As few students as possible should have more than one exam on the same day.
- ☐ Exams of an individual student should be distributed as evenly as possible over the examination session.
- ☐ The exams are distributed as evenly as possible over the examination period regarding the grading time each examiner requires.
- ☐ Exams should not take place during the lunch break (12:30 to 12:50).
- ☐ Exams should take place in a single room if possible.

Additional requirements:

- ☑ Exams take only place on week days.
- ☐ Exams take only place within the daily examination time.
- ☐ Some exams need to be fixed to a specific time slot and room.
- ☐ Examiners should have the possibility to block certain time slots.
- ☐ Rooms can be blocked during certain time slots.
- ☐ Exams should be split up into multiple rooms if the capacity of a room is too small.

An agile project development approach is applied to allow the customer to add additional requirements later on in the process. As a result, the customer always stays in the loop. If necessary, the customer then can intervene if something needs to be changed or was forgotten in the initial specification.

Chapter 3

Research

3.1 Cost Function for Optimal Distribution of Exams

Half of the soft constraints are related to an optimal distribution of exams. Therefore, we need a cost function that indicates how well a set of exams is distributed. This section describes the development and verification of the cost function and the reasoning behind it. As this cost function is crucial for a good exam scheduler, we tested it thoroughly before implementing the constraints based on it. The complete test results of the verification can be found in the Appendix C.

For the sake of simplicity, we define a few concepts that are heavily used in this section and form the basis of the cost function:

Definition 1. *Each exam has a timestamp that indicates on what date and what time the exam starts. A set of exams is called an examination schedule.*

Definition 2. *The distance between two exams is the absolute difference of the exams' timestamp. It can be measured in any unit of time. The duration of the exams is ignored.*

Definition 3. *Each examination schedule with $N \geqslant 2$ exams can be represented by a set of distances that contains all distances between two successive exams. The exams' order implied by the term "successive" is given by the exam's timestamp.*

For instance, the examination schedule E with three exams can be represented by the set D containing two distance, measured in hours:

$E = \{$ `"2021-01-19T08:00"` , `"2021-01-20T08:00"` , `"2021-01-22T13:00"` $\}$

$D = \{24, 53\}$

3.1.1 Initial Approach

As a first approach, we used the variance of all distances between two successive exams. However, we took the optimal distance as the mean. Let D be the distances representing a given examination schedule, and T is the exam session's duration in days. If $|D| \geqslant 1$, then we can calculate the optimal distance μ and the costs σ^2 for the given examination schedule. Note that we measured the distance in days. More precisely, we took only the absolute difference of the exam's date and truncated the time information.

$$\mu = \frac{T-1}{|D|} \tag{3.1}$$

$$\sigma^2 = \frac{1}{|D|} \sum_{d \in D} (d-\mu)^2 \tag{3.2}$$

The test results in Figure 3.1 reveal that the initial cost function is not normalized. This is a problem as we need to sum up the costs of multiple schedules. In that event, each examination schedule should be equally weighted. Otherwise, exam timetables with much higher costs have an enormous impact, which is not desirable. To solve this issue, we need to normalize the cost function.

Distances in hours		Distances in days		Description	Examination Schedule																				
Optimal D. [h]	Costs	Optimal D. [d]	Costs																						
72.00	419.87	3.00	0.67	7 Good	x			x				x		x		x				x			x		
72.00	1979.10	3.00	3.33	7 No free days	x	x	x	x	x			x	x												
72.00	1293.17	3.00	2.83	7 Cluster	x	x			x				x	x					x	x					
72.00	569.20	3.00	1.00	7 Compact but good	x		x		x			x			x	x			x						
39.27	182.39	1.64	0.60	12 Good	x	x		x	x			x	x		x	x			x	x		x	x		
39.27	471.32	1.64	1.01	12 Bad I		x	x	x	x			x	x	x	x				x	x	x	x			
39.27	915.48	1.64	1.87	12 Bad II	x	x	x	x					x	x	x	x				x	x	x	x		
432.00	46.69	18.00	0.00	2 Max spread	x																				x
432.00	160934.69	18.00	289.00	2 No spread				x	x																
432.00	129600.00	18.00	225.00	2 Good	x			x																	
108.00	2819.49	4.50	4.75	5 Compact but good	x		x					x		x		x									
108.00	245.49	4.50	0.25	5 Evenly spread all over	x				x					x					x						x
0.00	0.00	0.00	0.00	1 Exam								x													
30.86	282.12	1.29	0.49	15 Exams	x	x	x	x	x			x	x	x	x	x			x	x	x	x	x		

x Exam scheduled No exam scheduled Weekend

Figure 3.1: Test Results of the Initial Cost Function: It shows the costs and the optimal distance for different examination schedules, once measured in hours and once in days.

During the testing, we noticed that it is better to measure the distances between exams in hours rather than in days. If an examination schedule has a certain amount of exams, then it becomes inevitable that two exams take place on two consecutive days. At this point, the exam's start time becomes increasingly important for an even distribution of exams. Measuring the distances in hours considers the starting time and therefore results in better distinguishable scores, even for exam timetables with many exams. Moreover, we

realized that the optimal distance μ is not realistic. For instance, the schedule “12 Good” in Figure 3.1 has an optimal distance of about 40 h. However, a break so long between every exam is impossible. The optimal distances are too large because we include the weekends in calculating the optimal distance, but no exams can take place on weekends.

3.1.2 Improving the Cost Function in Four Iterations

1st Iteration: Normalization and Correction of Optimal Distance

The initial cost function (3.2) is in a way the variance of the distances $D = \{d_1, ..., d_n\}$ that represent an examination schedule. Consequently, we can use the coefficient of variation to normalize the cost function. The coefficient of variation v is defined as the ratio of the standard deviation σ to the mean μ. Since all distances are positive, we can define a normalized coefficient of variation v^* such that $0 \leqslant v^* \leqslant 1$ [3].

$$v = \frac{\sigma}{\mu} = \frac{\sqrt{\frac{1}{n}\sum_{d \in D}(d-\mu)^2}}{\mu} \tag{3.3}$$

$$v^* = \frac{v}{\sqrt{n}} = \frac{1}{n \cdot \mu}\sqrt{\sum_{d \in D}(d-\mu)^2} \tag{3.4}$$

The second problem is that we have unrealistic values for the optimal distances. To solve this issue, we simply ignored the days on which no exams can be scheduled when calculating the optimal distance (3.5). All of this together results in the updated cost function:

Let D be the distances in hours representing a given examination schedule with N exams, and T is the number of days on which exams can be scheduled. If $|D| \geqslant 1$, then we can calculate the optimal distance $\bar{d}$ between two successive exams and the costs v^* for the given examination schedule.

$$\bar{d} = 24 \cdot \frac{T-1}{N} \tag{3.5}$$

$$v^* = \frac{1}{|D| \cdot \bar{d}}\sqrt{\sum_{d \in D}(d-\bar{d})^2} \tag{3.6}$$

Figure 3.2 shows two unexpected behaviors of the updated cost function. We expected all costs to be in the range $[0; 1]$. However, not all costs seem to be properly normalized. Even more surprisingly, the costs for the examination schedule “7 Good” are now higher than the cost for the examination schedule “7 No free days”.

Intial Aproach		1st Iteration		Description	Examination Schedule																					
Optimal D. [h]	Costs	Optimal D. [h]	Costs																							
72.00	419.87	48.00	0.267	7 Good	x			x				x		x		x				x			x	
72.00	1979.10	48.00	0.210	7 No free days	x	x	x	x	x			x	x											
72.00	1293.17	48.00	0.313	7 Cluster	x	x			x				x	x					x	x				
72.00	569.20	48.00	0.163	7 Compact but good	x		x		x			x			x	x			x					
39.27	182.39	28.00	0.193	12 Good	x	x		x	x			x	x		x	x			x	x		x	x
39.27	471.32	28.00	0.244	12 Bad I		x	x	x	x			x	x	x	x				x	x	x	x		
39.27	915.48	28.00	0.350	12 Bad II	x	x	x	x					x	x	x	x				x	x	x	x	
432.00	46.69	168.00	1.612	2 Max spread	x																			x
432.00	160934.69	168.00	0.816	2 No spread				x	x															
432.00	129600.00	168.00	0.571	2 Good	x			x																
108.00	2819.49	67.20	0.232	5 Compact but good	x		x					x		x		x								
108.00	245.49	67.20	0.318	5 Evenly spread all over	x				x					x					x					x
0.00	0.00	0.00	0.000	1 Exam								x												
30.86	282.12	22.40	0.224	15 Exams	x	x	x	x	x			x	x	x	x	x			x	x	x	x	x

x Exam scheduled No exam scheduled Weekend

Figure 3.2: Comparison of the Cost Function's Results (Initial Approach vs. 1st Iteration)

2nd Iteration: Ignoring Distances Greater than the Optimal Distance

So far, the cost function aims to distribute the exams as evenly as possible. This goal seems wrong, though, as most students would not mind if exams are farther apart than the optimal distance. Hence, it seems natural to ignore all distances that are greater than the optimal distance.

The reason that some costs are not normalized as expected is due to the fact that we decreased the optimal distance, and so the enumerator in the cost function (3.6) gets too small. Fortunately, this normalization problem goes away when the distances greater than the optimal distance are ignored.

1st Iteration		2nd Iteration		Description	Examination Schedule																					
Optimal D. [h]	Costs	Optimal D. [h]	Costs																							
48.00	0.267	48.00	0.025	7 Good	x			x				x		x		x				x			x	
48.00	0.210	48.00	0.193	7 No free days	x	x	x	x	x			x	x											
48.00	0.313	48.00	0.111	7 Cluster	x	x			x				x	x					x	x				
48.00	0.163	48.00	0.073	7 Compact but good	x		x		x			x			x	x			x					
28.00	0.193	28.00	0.000	12 Good	x	x		x	x			x	x		x	x			x	x		x	x
28.00	0.244	28.00	0.055	12 Bad I		x	x	x	x			x	x	x	x				x	x	x	x		
28.00	0.350	28.00	0.055	12 Bad II	x	x	x	x					x	x	x	x				x	x	x	x	
168.00	1.612	168.00	0.000	2 Max spread	x																			x
168.00	0.816	168.00	0.816	2 No spread				x	x															
168.00	0.571	168.00	0.571	2 Good	x			x																
67.20	0.232	67.20	0.133	5 Compact but good	x		x					x		x		x								
67.20	0.318	67.20	0.000	5 Evenly spread all over	x				x					x					x					x
0.00	0.000	0.00	0.000	1 Exam								x												
22.40	0.224	22.40	0.000	15 Exams	x	x	x	x	x			x	x	x	x	x			x	x	x	x	x

x Exam scheduled No exam scheduled Weekend

Figure 3.3: Comparison of the Cost Function's Results (1st Iteration vs. 2nd Iteration)

At first glance, the test results from the second iteration, shown in Figure 3.3, looks good. Bad examination schedules have higher costs than good ones with the same amount of exams. However, the costs of examination schedules with fewer exams tend to be higher in general, which is a new normalization problem.

For example, the examination schedule “2 Good” in Figure 3.3 is quite acceptable, but its costs of 0.571 are ten times higher than the costs of the schedule “12 Bad I”. The costs of exam timetables should be comparable independent of the number of exams.

3rd Iteration: Max Value for the Optimal Distance

Exam timetables with fewer exams have a higher optimal distance since the cost function aims to distribute the exams as evenly as possible. The examination schedule “2 Good” in Figure 3.3 has an optimal distance of 168 h, which is a lot. In our opinion, three nights between two exams is sufficiently good. Therefore, we set a max value of 60 h for the optimal distance. Note that we set it to 60 h and not 72 h to ensure that the max number of nights between two exams is always exactly three. With 72 h, it would be four nights if the first exam takes place in the afternoon.

With the upper limit for the optimal distance, we aimed to make examination schedules with a different number of exams comparable. However, the third iteration test results, shown in Figure 3.4, tell us that we achieved the desired goal only partly. The costs for the schedule “2 Good” are now zero, and the costs for the schedules “2 No spread” and “5 Compact but good” decreased in comparison to the second iteration. Nevertheless, all other costs did not change at all because the upper limit of the optimal distance affects only the costs of examination schedules whose actual optimal distance is below the limit.

2nd Iteration		3rd Iteration		Description	Examination Schedule																					
Optimal D. [h]	Costs	Optimal D. [h]	Costs																							
48.00	0.025	48.00	0.025	7 Good	x			x				x		x		x				x				x		
48.00	0.193	48.00	0.193	7 No free days	x	x	x	x	x			x	x													
48.00	0.111	48.00	0.111	7 Cluster	x	x			x				x	x					x	x						
48.00	0.073	48.00	0.073	7 Compact but good	x		x		x			x			x	x			x							
28.00	0.000	28.00	0.000	12 Good	x	x		x	x			x	x		x	x			x	x			x	x		
28.00	0.055	28.00	0.055	12 Bad I		x	x	x	x			x	x	x	x				x	x	x	x				
28.00	0.055	28.00	0.055	12 Bad II	x	x	x	x					x	x	x	x				x	x	x	x	x		
168.00	0.000	60.00	0.000	2 Max spread	x																					x
168.00	0.816	60.00	0.486	2 No spread				x	x																	
168.00	0.571	60.00	0.000	2 Good	x			x																		
67.20	0.133	60.00	0.100	5 Compact but good	x		x					x		x		x										
67.20	0.000	60.00	0.000	5 Evenly spread all over	x				x					x					x							x
0.00	0.000	0.00	0.000	1 Exam								x														
22.40	0.000	22.40	0.000	15 Exams	x	x	x	x	x			x	x	x	x	x			x	x	x	x	x			

x Exam scheduled No exam scheduled Weekend

Figure 3.4: Comparison of the Cost Function’s Results (2nd Iteration vs. 3rd Iteration)

Iteration 4: Weighing the Costs According to the Number of Exams

Still, examination schedules with fewer exams have lower costs in general. Ideally, all schedules with an optimal exam distribution have costs close to zero, and a worse distribution should result in higher costs. Moreover, two examination schedules with an equally good distribution should have similar costs, independent of the number of exams. From

the previous iterations, we know that distributing the exams optimally becomes more difficult with more exams. Technically, it gets more challenging as the number of days on which exams can be scheduled is limited. Therefore, we came up with the idea of adding a weighting according to the ratio of the number of exams to the available time. With this adjustment, we can define the final version of the cost function.

Definition 4. *Let D be the distances in hours representing a given examination schedule with N exams, and T the number of days on which exams can be scheduled. If $|D| \geqslant 1$, the optimal distance $\bar{d}$ between two successive exams and the costs v^* for the given examination schedule can be calculated.*

$$\bar{d} = \min(24 \cdot \frac{T-1}{N}, 60) \tag{3.7}$$

$$D^* = \{d \in D | d \leqslant \bar{d}\} \tag{3.8}$$

$$v^* = \frac{1}{|D| \cdot \bar{d}} \sqrt{\sum_{d \in D^*} (d - \bar{d})^2} \cdot \frac{N}{T} \tag{3.9}$$

For the verification, we created examination schedules with different numbers of exams and put them in one of the three categories:

1. “good”
2. “bad”
3. “worst”

The exact definition of these categories is highly subjective and therefore not that relevant. More importantly, though, is its order. For a group of examination schedules with the same number of exams, we expect the schedules of the category “good” to have the lowest score and the schedules of the category “worst” to have the highest score. The score of the examination schedule with the category “bad” should be somewhere between. Besides, the schedules of a the same category should have similar scores.

The column “4th Iteration” in Figure 3.5 shows the costs from the final cost function. For a better comparison, the costs are multiplied by 1000. For a group of schedules with the same amount of exams, the score ensures the same order as stated by the categories. One can also see that schedules from the same category have a similar score:
“good” $\rightarrow [0:8]$, “bad” $\rightarrow [23:58]$, “worst” $\rightarrow [66:100]$

Costs		Examination Schedule
3rd Iteration	4th Iteration	
0	0	3 Exams (good)
141	28	3 Exams (bad)
424	85	3 Exams (worst)
0	0	4 Exams (good)
94	25	4 Exams (bad)
346	92	4 Exams (worst)
0	0	5 Exams (good)
87	29	5 Exams (bad)
300	100	5 Exams (worst)
0	0	6 Exams (good)
57	23	6 Exams (bad)
229	91	6 Exams (worst)
24	11	7 Exams (good)
83	39	7 Exams (bad)
167	78	7 Exams (worst)
0	0	8 Exams (good)
87	46	8 Exams (bad)
150	80	8 Exams (worst)
0	0	9 Exams (good)
89	54	9 Exams (bad)
118	71	9 Exams (worst)
12	8	10 Exams (good)
84	56	10 Exams (bad)
119	79	10 Exams (worst)
0	0	11 Exams (good)
79	58	11 Exams (bad)
94	69	11 Exams (worst)
0	0	12 Exams (good)
45	36	12 Exams (bad)
82	66	12 Exams (worst)

Figure 3.5: Comparison of the Cost Function’s Results (3rd Iteration vs. 4th Iteration)

3.2 Preparing the Problem for a Constraint Solver

When trying to find a feasible solution for an examination schedule with a constraint planner such as the OptaPlanner, one can do many optimizations. These optimizations can either be adjustments of the problem model or different choices of algorithms. This section identifies the problem's search space and explains what kind of solution can be found at all. Moreover, it talks about how the OptaPlanner works internally and preparation work needed for an efficient solving process.

3.2.1 Identifying the Search Space

Creating an examination schedule is an NP-complete problem [1]. To optimize the solving process, we need to understand the search space and check what restrictions can reduce it. To identify the search space, we take an actual input data set. This set has:

- 196 exams
- 20 rooms
- 405 possible time slots of 20 minutes each (15 days, 08:00 – 17:00)
- 1,322 students
- 9,139 exam registrations

If we only take a single exam into account, we have 405 time slots times 20 rooms resulting in 8,100 possible configurations.

$$\text{timeSlots} \cdot \text{rooms} = \text{possibleConfigurationsForASingleExam} \tag{3.10}$$

Note that we simplify the problem by assuming each exam takes place in precisely one room. 8100 is already a significant number, but it would be attainable.

When looking at the entire data set, everything changes. Let T be a timeline with the same number of slots as exams exist. We can now put every exam in any order in every position. If we do this for each combination, we end up with $196!$ different configurations for T. $196!$ is somewhere in the neighborhood of $5.08 \cdot 10^{365}$, which is a massive number of options. In comparison, the number of atoms in the universe is only somewhere between 10^{78} and 10^{82}. Furthermore, do not forget that the different possible time slots, and the fact that some exams take place in multiple rooms are excluded here.

With the above-made calculations, we now have a rough understanding of how ample the initial solution space is. In the following sections, we investigate how this solution space can be reduced and what options we have to find a viable solution in a reasonable time.

3.2.2 Can we Find an Optimal Solution?

Based on the previously seen solution space, the question arises if we, or the OptaPlanner, can find an optimal solution. Since the problem is NP-complete, the short answer is no, most certainly not. The OptaPlanner team also tackled this question in their documentation and noted what other requirements there might be and what influence they have:

“

- Scale out: Large production data sets must not crash and have also good results.
- Optimize the right problem: The constraints must match the actual business needs.
- Available time: The solution must be found in time, before it becomes useless to execute.
- Reliability: Every data set must have at least a decent result (better than a human planner).

Given these requirements [...], it is usually impossible for anyone or anything to find the optimal solution. Therefore, OptaPlanner focuses on finding the best solution in available time. [...] The nature of NP-complete problems make scaling a prime concern. ” ([4])

Taking the huge solution space seen in the previous section, we know that there is no way for the OptaPlanner to find an optimal solution. Therefore, the OptaPlanner uses different algorithms, attempting to find a feasible solution in an acceptable time. We can configure these algorithms to work best with our problem.

Before introducing the algorithms and its configurations, we look at some preparation work that enables the algorithms to work as efficiently as possible.

3.2.3 Preparation Work

The OptaPlanner can do a lot on its own as soon as the problem model is defined. Nevertheless, there are many ways to help it perform more efficiently.

Reduce Problem Facts per Planning Entity

As a short recap; Planning entities are, as the name might suggest, the entities that need to be planned. Problem facts are the possible values that can be assigned to a planning

entity. In the case of the exam scheduler, the exams are the planning entities and the rooms and time slots are the problem facts. In literature, the problem facts are also called the *domain*, that gets assigned to a set of *variables* on the planning entity [5].

Without any particular configuration, every planning entity has the same problem facts at its disposal. As seen in subsection 3.2.1, this results in 8,100 possible configurations per exam. If we have some a priori knowledge, we can reduce this number by quite a bit.

For example, an exam that has 50 students needs a room with at least that capacity. Therefore, we can remove all rooms that have a smaller capacity for this exam. In our example, this would mean that we would go from 20 rooms down to three rooms. Applying the previous calculation (Equation 3.10), we get 1,215 possible configurations. This result is 85% less than what we had before. Suppose we now also remove all the timeslots in the evening that are not a possible start time, as the exam would end after “work hours”. In that case, we can reduce the possible solutions even further down to 945, which is only 12% of the initial solution space.

With the help of this preparation work, we can keep the OptaPlanner from testing assignments that never result in a viable solution.

Selection Filtering During Runtime

Some assignments may be initially possible but turn out to be not viable based on the solution state present. For situations like this, we have the option to add selection filters. These filters take a selection and the current solution state and return whether the selection is accepted or not. For example, a selection can be the assignment (move) of a new room or time slot to an exam. If this selection is not acceptable based on the current solution state, because the new room has not enough space, the filter will remove this selection and even further reduce the solution space.

Weighting the Constraints

The exam scheduler aims to find an examination schedule that fulfills all constraints listed in section 2.3. For a feasible solution, all hard constraints *must be* fulfilled. On the other hand, soft constraints *should be* fulfilled but can be broken if needed. So, overall hard constraints have a higher priority than soft constraints. Moreover, the soft constraints are prioritized amongst each other. Therefore, some constraints need to be weighted more than others. Unfortunately, one can predict the impact of a weighting factor only to some extent. This leads to the fact that the first try most likely will not be what we wanted. That is why an initial educated guess of weights has to be made, which can be iteratively adjusted to meet the expectations. In the end, the constraints are more or less single-

handedly responsible for the quality of the solution. No matter what algorithms are used, the solution will be bad if the constraints are poorly weighted.

Providing Additional Information about the Problem Model

To help a constraint solver fulfill its task, we can give it some extra information about the problem model. The OptaPlanner gives us the possibility to specify the behavior of the score resulting from the constraints via the `initializingScoreTrend` option [6]. For example, if we only have negative scores, we know that when assigning a value to a previously unassigned variable, the score can only stay the same or get worse but never get better. With this extra information, the solver gains many insights into the model. If a configuration results in a score equal to the previous configuration score, the solver can stop looking. If it would not have this information, every configuration had to be tested to find the configuration which results in the best score.

Another possibility to help the OptaPlanner, respectively the underlying algorithms, is to implement a `difficultyComparatorClass` [7]. This class defines a planning difficulty for each planning entity. In the case of the exam scheduler, this difficulty could be, for example, the number of students an exam has, the duration of the exam, or a combination of both. With the help of this information, the OptaPlanner can start with the assignment of the complex entities and focus on the "easier/simpler" ones later on.

Tweak Calculation Speed

Even with all the preparation work done, the process will only be as fast as the slowest component. We have to make sure that this is not our code. With every changing variable, all constraints are checked, and with them, all methods responsible for calculating the score are executed. Any ineffective calculation in our code will immensely slow down the solving process. Therefore, special attention needs to be paid that the calculations are done efficiently. However, this is not the end. Optimizing the efficiency of the calculation itself is not enough. Whenever possible, additional caching should be used, especially if the calculations are executed many times with the same parameters.

3.3 Algorithm-Evaluation for the OptaPlanner

This section evaluates the different types of algorithms available for "solving" NP-complete problems with the OptaPlanner. For the most promising algorithms, we go into depth on how they work and how they compare to each other.

3.3.1 Exhaustive Search

Exhaustive search is the most simple kind of algorithms. As the name suggests, these algorithms exhaust all possible solutions. As we discussed in subsection 3.2.1, there are $n!$ possible solutions and so the time complexity is $O(n!)$. The algorithms will find the best solution eventually but, unless the problem space is small, not in a matter of time that any human being will ever be able to witness it.

At this point, we introduce two properties of an algorithm. Being *complete*, meaning that the algorithm always finds a solution in a finite amount of time if one exists, and being *optimal*, meaning that the algorithm always finds a global minimum/maximum [5].

Even though exhaustive search algorithms, such as brute force or branch and bound, are complete and optimal, the time complexity is too bad. So, this approach can be discarded right from the beginning.

3.3.2 Solving Process and its Phases

The OptaPlanner solving process consists of multiple phases. Each of them uses different algorithms. Usually, the first phase is the construction phase.

Construction Phase

In the construction phase, the task is to take an empty or partially assigned solution and initialize all the variables with a value. Th OptaPlanner does this with the help of some construction heuristics [8]. Construction heuristics do this job of assigning values to the variables in a finite length of time. The resulting solution might violate hard constraints, but it is found fast, optimally, in less than half a minute if the problem scale allows it [9]. The job of the local search or evolutionary phase is then to start improving this initial solution.

Local Search Phase

The local search phase always needs an initialized solution. That is why the construction phase is executed preliminary. The local search phase runs, as the name suggests, one or multiple local search algorithms.

Local search algorithms belong to the family of Metaheuristics. Sean Luke answers the question “What is a Metaheuristics?” in his book “Essentials of Metaheuristics” as follows:

> A common but unfortunate name for any stochastic optimization algorithm intended to be the last resort before giving up and using random or brute-force search. Such algorithms are used for problems where you don’t know how to find a good solution, but if shown a candidate solution, you can give it a grade. ([10])

For visualizing this concept, imagine “painting the Mona Lisa” is the problem and copy printers are not available. Doing this, will take a tremendous amount of time, so we ease the problem by only requiring that it needs to be recognizable by a human. With only 45 polygons layered on top of each other, one can create a rather impressive result, shown in Figure 3.6.

Figure 3.6: Analogy of an Algorithm Trying to Find a Solution in Reasonable Time [10]

Local search algorithms aim to find the best possible solution, which means maximizing the score or minimizing the penalty, respectively. The main handicap of such algorithms is that they may get stuck in a local maxima/minima. Local maxima/minima are solutions where better solutions exist, but slightly changing the current solution always results in a worse solution. There are many different algorithms, the main difference being how they handle such situations. As opposed to a local maxima/minima, we have the global maxima/minima, representing the best possible solution.

As we have seen in subsection 3.2.1, the number of possible configurations (called *moves*) for each state (called *step*) can get large. Thus, this number has to be limited in order for the algorithm not to move too slowly. For that, we have to configure the possible-moves-per-step forager of the OptaPlanner with a limit, namely the `acceptedCountLimit` [11]. Setting the limit very low results in a quick-stepping algorithm, meaning that only a few moves are taken into account before selecting one. This has the drawback of not always choosing the best option. Setting the limit very high, i.e. take a lot of moves into account, improves the solution pick but decreases the step speed. This value has to be chosen carefully depending on the selected algorithm. According to the OptaPlanner documentation, each step should take less than two seconds as a general guideline [12].

Evolutionary Phase

The evolutionary phase consists of one or multiple evolutionary algorithms. Evolutionary algorithms also belong to the family of Metaheuristics. They operate on a population and generate new states by mutating and crossing over those populations. We will not go into details on those, as first, this type of algorithms does not suit our needs, and second, OptaPlanner has, at the time of writing this, no support for them.

3.3.3 Algorithms in Construction Phase

This section talks about the algorithms used in the construction phase. The focus lies on one particular algorithm, but alternatives and a variant are discussed as well.

First Fit Algorithm

The OptaPlanner uses in the default configuration the first fit algorithm. This algorithm goes through all the planning entities one by one and assigns the best planning value(s) available to it. Once a planning value is set, it is never changed again. The algorithm ends when all planning entities are initialized.

Alternative Algorithms

The OptaPlanner also has some other algorithms available: weakest fit, strongest fit, allocate entity from queue, allocate to value from queue, cheapest insertion, regret insertion, and allocate from pool. We will not detail them, as they do not promise any improvement over the first fit algorithm. The OptaPlanner documentation recommends using the first fit algorithm for most cases.

Decreasing Variants

A unique variation that can be applied to most named algorithms is the “decreasing” version. In this version, the algorithm uses a list of exams that are sorted by their planning difficulty. The difficulty is specified on the planning entity as described in section 3.2.3.

Using the first fit decreasing variant, the algorithm assigns the more difficult planning entities first and ends with the “easy” ones.

3.3.4 Algorithms in Local Search Phase

This section talks about algorithms used in the local search phase and what relations they have to each other (see Figure 3.7).

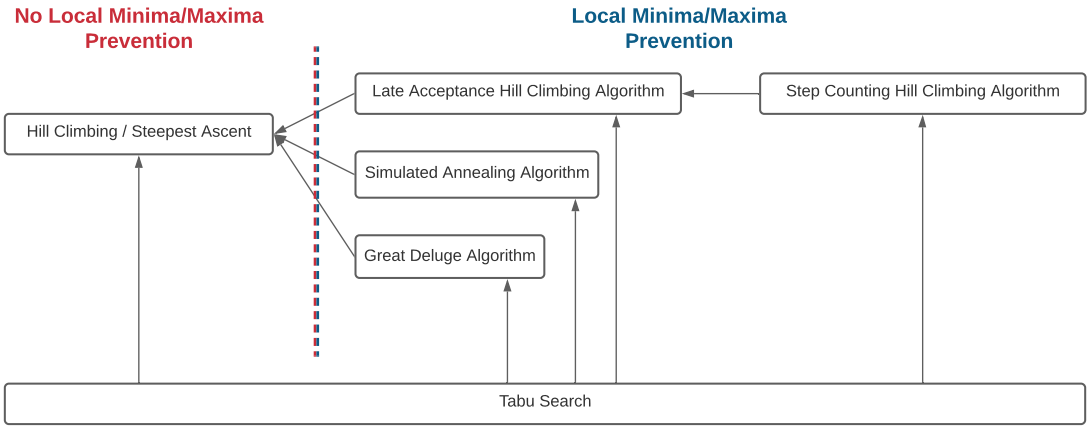


Figure 3.7: Metaheuristc Map with Dependencies

Hill Climbing/Steepest Ascent

The hill climbing algorithm is the simplest variant of a local search algorithm. It works by taking the current state, adjusts a few variables, and compares the newly resulted score to the score of the current state. If the score improves, this new solution gets the new state, the old state gets discarded, and everything starts again. This algorithm is very memory efficient as no knowledge about previous states is stored. However, if this algorithm lands in a local maximum, it has no chance of ever getting out of it again. Stuart Russell and Peter Norvig nicely explain this in their book “Artificial Intelligence - A Modern Approach” [5] as being the same as “trying to find the top of Mount Everest in a thick fog while suffering from amnesia.”.

For the hill climbing algorithm to always choose the steepest possible improvement, the number of possible next-moves, `acceptedCountLimit`, must not be limited too much. Hill climbing is neither complete nor optimal with the time complexity of $O(\infty)$ [13].

Tabu Search Algorithm

The tabu search algorithm improves the hill climbing algorithm by maintaining a list of recently visited states and forbids the algorithm to choose them again. This list has a fixed size and is updated with every move [14, 15].

To explain tabu search, let us apply it to an example. We can think of local search as being trapped in a maze. Every corner and hallway looks the same. This indistinguishability will lead to cases where we visit the same spot over and over again without ever noticing it. Now to help us navigate, we tie a rope around our waist and pull it behind us. Whenever we come across a part of the rope, we know we have been here before. This rope represents the tabu list in our search algorithm. To trace the whole path we took, we would need a rope that is of infinite length (this represents the space complexity of the problem). Therefore, we need to cut it into a finite length. With every step we move forward, the end also moves, and we lose some information about our path. Similar to the arcade game “Snake”. The main challenge now is to choose the right length of the rope (the tabu list) [16].

The OptaPlanner calls this list length/size `...TabuSize` or `...TabuRatio`, where the tabu list can be applied to (planning) entities, problem facts (called *values* in the OptaPlanner configuration), moves, or combinations of all the three listed. Together with the list size, same as in the hill climbing algorithm, the `acceptedCountLimit`, must not be limited too hard. Else, it even can happen that the algorithm gets stuck in places where all available moves are in the tabu list and are therefore forbidden to choose.

Tabu search is not limited to the hill climbing algorithm. It can be applied to every local search algorithm.

Simulated Annealing Algorithm

Simulated annealing, or short SA, is a more advanced local search algorithm and is kind of a combination of random walk (making every move completely random) and hill climbing. It tries to solve the problem of getting stuck in a local maxima/minima by accepting some moves that temporarily worsen the solution. Annealing comes from metallurgy and is the process of tempering or hardening metals or glass by heating them up and then letting them slowly cool off. Stuart Russell and Peter Norvig describe this as follows:

> To explain simulated annealing, we switch our point of view from hill climbing to gradient descent (i.e., minimizing cost) and imagine the task of getting a ping-pong ball into the deepest crevice in a bumpy surface. If we just let the ball roll, it will come to rest at a local minimum. If we shake the surface, we can bounce the ball out of the local minimum. The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum. The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and then gradually reduce the intensity of the shaking (i.e., lower the temperature). ([5])

Simulated annealing is a quite fast converging algorithm. By default, the first accepted move is also the winning step. Therefore, the `acceptedCountLimit` should be kept small. Accepted means that it either improves the score or passes a random check. If we go back to the ping-pong ball example, the random check is a quick shake, and the probability of selecting a "bad" move is relative to the shaking strength (the temperature).

When configuring the simulated annealing approach in the OptaPlanner, an initial maximum temperature (maximum shaking strength) is set. Throughout the solving process, this temperature is reduced.

To prevent the algorithm from changing the same variables repeatedly without any real improvement, simulated annealing can also be combined with the tabu search. However, the tabu list size should be kept smaller than when using the tabu search with hill climbing algorithm, as the `acceptedCountLimit` is also small.

Simulated annealing is not complete but gets close to being optimal. The worst-case scenario is still a time complexity of $O(\infty)$ as it is not guaranteed to find a global maximum/minimum. Nevertheless, in most cases, it performs much better than the hill climbing algorithm [13].

Late Acceptance Hill Climbing Algorithm

Late acceptance hill climbing, or short LAHC, is a specialization of the hill climbing algorithm discussed before. Instead of comparing the new state with the current state, late acceptance hill climbing compares the new state to the state a number of steps back, namely `lateAcceptanceSize`. This late acceptance tries to prevent it from ending up in local minima/maxima. Similar to other algorithms before, the real challenge lies in configuring the algorithm the right way. Here the number of steps back has to be configured. If chosen too low, we end up with the default hill climbing algorithm (`lateAcceptanceSize = 0`). If chosen too high, we end up with an algorithm that accepts every move and is therefore

just a random walk. As this algorithm is also open for change, the `acceptedCountLimit` is set low in order for it to move fast.

Based on two studies, late acceptance hill climbing often performs better by a small margin than simulated annealing on the problem of scheduling exams [17, 18].

Great Deluge Algorithm

Great deluge is very similar to simulated annealing. Instead of a temperature, it uses a water level. The water level represents the minimal score a new solution needs to have to be accepted. The first solution that is accepted is also chosen. After each step, the water level is increased. As the level increases gradually, this algorithm has more time to escape from a local maxima/minima.

Figure 3.8 demonstrates this behavior. The start point is chosen randomly. In step two, the algorithm has found a local maxima, but because the water level is low enough, the solution can go down hill (get worse) temporarily, for it to find a new ascending. This goes on for the next steps until it finds the global maxima, i.e. the optimal solution, where it can not go anywhere else. If the water level surpasses the best solution found, the search is over. Finding the optimal solution/the goal state, is not guaranteed, but the chances are much higher than with the hill climbing algorithm.

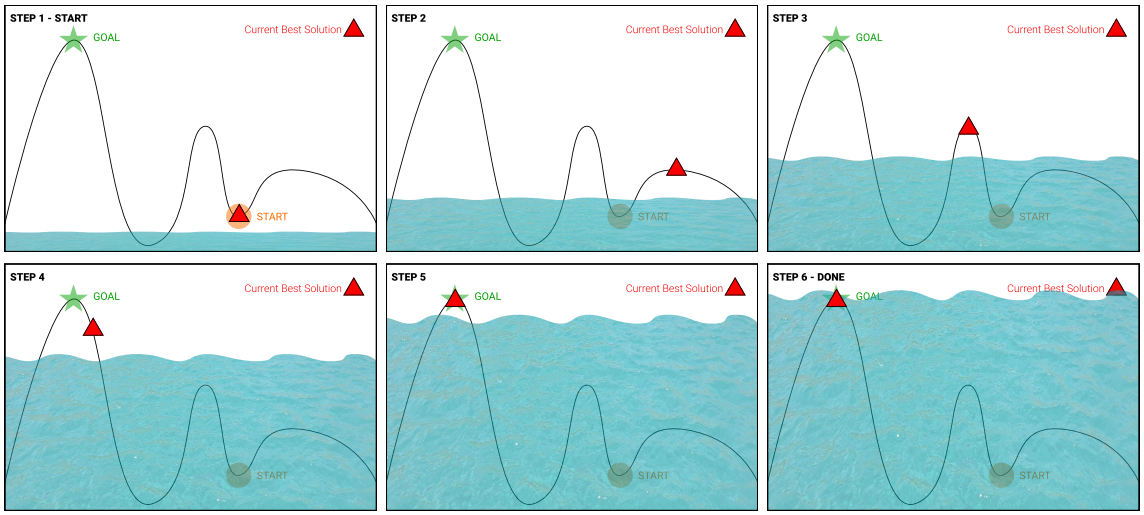


Figure 3.8: Process of the Great Deluge Algorithm Finding the Optimal Solution

As with any other algorithm discussed, finding a suitable configuration is the challenge. In the case of great deluge, the initial water level (`greatDelugeInitialWaterLevel`) and the water level increment step (`greatDelugeWaterLevelIncrement`) needs to be configured. The `acceptedCountLimit` is set low for the same reasons as in simulated

annealing. Based on the same two studies mentioned above, great deluge performs worse than SA and LAHC [17, 18].

Step Counting Hill Climbing Algorithm

The step counting hill climbing algorithm, or short SCHC, is a variation of the late acceptance hill climbing algorithm. Instead of comparing the new score to the threshold score x moves before, SCHC selects a new threshold every x moves and keeps this one for the following x moves. A visualization of the difference can be seen in Figure 3.9.

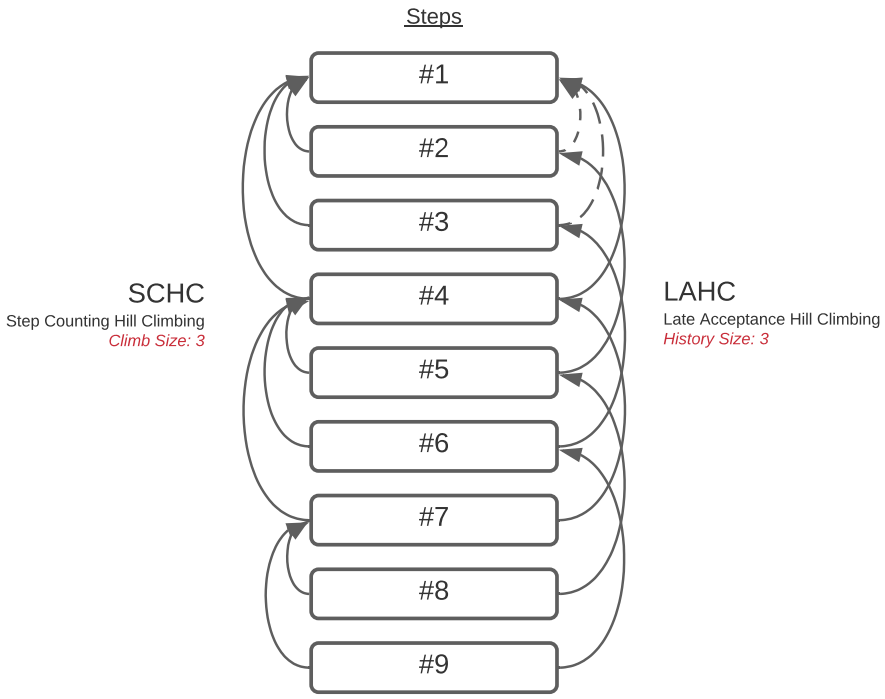


Figure 3.9: Comparison of the Internal Workings of Step Counting Hill Climbing and Late Acceptance Hill Climbing

For this version, the number of steps, namely the `stepCountingHillClimbingSize`, have to be set. Everything else is identical to LAHC. Based on the same two studies mentioned above [17, 18], SCHC performs even better than LAHC.

3.3.5 Conclusion and Configurations Short List

A significant chunk of optimizations can be done by preparing the data model and supplying the OptaPlanner with additional information regarding the model. Those optimizations

should be the first order of action; Adding the `difficultyComparatorClass` and an `initializingScoreTrend`.

After that, suitable algorithms have to be chosen. We only discussed the two most promising variants for the construction phase, which we are both going to test.

- First Fit Algorithm
- First Fit Decreasing (Decreasing Variants)

For the local search phase, more different options were covered. Based on the found research and the suggestions of the OptaPlanner documentation, we are going to test:

- Simulated Annealing Algorithm
- Step Counting Hill Climbing Algorithm

The better configuration of both should also be tested in a second round, where a bit of Tabu should be included. Furthermore, some tests regarding the `acceptedCountLimit` have to be made to find a suitable configuration.

Chapter 4

Solution

This chapter describes all the enhancements and adjustments to the previous version of the exam scheduler resulting from the semester project. It also tackles details regarding performance optimizations, improvements of the UI, the integration into the university's IT system, and the quality assurance measures put in place.

4.1 Adjustments and Enhancements of the Exam Scheduler

This section explains in depth all the additional requirements that required some particular decisions and compromises.

4.1.1 Support for Multiple Rooms per Exam

User Story: D.1.6

Until now, the exam scheduler always assigned only one room to an exam. This strategy works well as long as every exam does not have more students than the biggest room has capacity. This assumption is not always fulfilled and also should not be enforced. Therefore, the exam scheduler needed to be adapted to meet this requirement.

One can think of many different approaches to solving this problem. To give just a few options:

1. Having a variable list of rooms for each exam that can be assigned
2. Creating virtual rooms that combine two or more rooms
3. Having a defined number of room slots for each exam that can be assigned

Option 1 is the most generic and flexible solution. The problem is that it is too flexible for the OptaPlanner to be able to work with it. Furthermore, it would increase the solution space exponentially for every additional possible room combination.

Option 2 keeps the solution space for the room assignment linear but would have way more possible rooms that can be assigned. This drawback would therefore increase the base size of the solution space. Besides that, it would be complicated to keep track of the room assignments inside those virtual rooms to prevent assigning the same room at the same time to different exams.

Option 3, which we used in the end, combines both approaches' benefits to some extent. Each exam has a limited number of slots where a room can be assigned to. In our case, we settled for four slots. When a room needs to be selected, all rooms except the already assigned ones are possible. Removing an assignment is also a viable option as long as at least one room is set for the exam. With this approach, the base solution space, meaning the number of possible rooms, is kept small, and the maximum size is limited. As a further improvement, we added the possibility to limit the number of room slots per exam. This limitation is beneficial for controlling when splitting an exam into two rooms is desired and when not. For example, an exam with 30 students should never be split up into two rooms with a capacity of 20 people, when there are plenty of other rooms that can easily contain 30 students. On the other hand, an exam with 240 students must be split into two or even more rooms if the largest room's capacity is insufficient.

For a human planner, these restrictions would suffice. However, the OptaPlanner works in ways that it assigns new problem facts to the planning variables and checks afterward if the score improved. If this is not the case, this solution is discarded. This way of solving the problem is acceptable for most constraints, but when assigning rooms, this led to instances where extra rooms were assigned that were not needed. For example, an exam with 80 students could end up with three rooms assigned with capacities of 50, 20, and 40, respectively. This extra assignment was not only a problem for the final output, but it was also a massive problem regarding finding an optimal solution. As long as a room was assigned to an exam at a particular time, it could not be used for other exams due to another hard constraint. A new constraint was needed that ensures that unused rooms are released for assignment to other exams.

4.1.2 Optimal Exam Distribution for Regular Semesters and Students

User Stories: D.1.7, D.1.8

The implementation of the cost function for an optimal exam distribution is straightforward. The methods `calculateOptimalDistanceBetweenTwoExamsInTimeGrains` and `calculateTotalCosts` in Listing 4.1 correspond to the formulas in Definition 4. However, using the cost function to build a constraint that ensures an optimal distribution of exams for students, or within a regular semester, was more complicated than it initially looked. The cost function indicates how well a set of exams is distributed overall. Hence, the cost value is a good measure for the quality of an examination schedule. But it gives

no information on how a single exam influences the costs. Giving the user the costs is like telling whether the examination schedule is good or bad, but without any reason for the cause. Ideally, the user can see which exams are responsible for a bad distribution. With the OptaPlanner's constraint stream API, the penalty for a bad distribution can be broken down to the individual exams.

```
1  public class OptimalExamDistributionCostFunction {
2     // ...
3     private double calculateOptimalDistanceBetweenTwoExamsInTimeGrains() {
4        if (numberOfExams < 2) {
5           return 0;
6        }
7        double optimalDistanceInMinutes = ((double) availableTimeForSchedulingInMinutes) / numberOfExams;
8        optimalDistanceInMinutes = Math.min(optimalDistanceInMinutes, MAX_OPTIMAL_DISTANCE_IN_MINUTES);
9        return optimalDistanceInMinutes;
10    }
11
12    private double calculateTotalCosts(List<Exam> sortedExams) {
13       if (sortedExams.size() < 2) {
14          return 0;
15       }
16       double[] distances = getTimeGrainDistancesOfSuccessiveExams(sortedExams);
17       // residual sum of squares
18       double rss = Arrays.stream(distances).parallel()
19             .filter(d -> d < optimalDistanceBetweenTwoExamsInTimeGrains)
20             .map(d -> (Math.pow(d - optimalDistanceBetweenTwoExamsInTimeGrains, 2)))
21             .sum();
22       return (numberOfExams) / (distances.length * optimalDistanceBetweenTwoExamsInTimeGrains * availableDaysForScheduling) * Math.sqrt(rss);
23    }
24    // ...
25 }
```

Listing 4.1: OptimalExamDistributionCostFunction.java

In our first attempt we divided the costs for a set of exams by the number of exams. Soon, we realized that this approach causes misleading information. A simple example best points out the problem. For simplicity, only the scores of a single student are considered. Figure 4.1 shows the student's examination schedule. A human can easily tell that the first week is stressful, whereas there are not many exams in the last week.

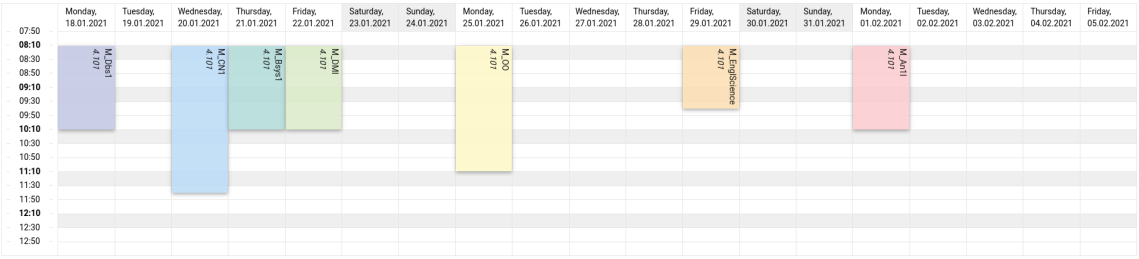


Figure 4.1: Sample Examination Schedule with Seven Exams for One Student

Figure 4.2 shows the breakdown of costs. Each exam is punished with a score of `-80soft` because the total costs are divided by the number of exams. This information seems wrong as the exams in the second and third weeks are well distributed. Only the first week is problematic.

Datenbanksysteme 1	0 hard / -80 soft	M_Dbs1	120 min
Betriebssysteme 1	0 hard / -80 soft	M_Bsys1	120 min
Objektorientierte Programmierung	0 hard / -80 soft	M_OO	180 min
Computernetze 1	0 hard / -80 soft	M_CN1	210 min
English: The World of Science	0 hard / -80 soft	M_EnglScience	90 min
Diskrete Mathematik für Informatik	0 hard / -80 soft	M_DMI	120 min
Analysis 1 für Informatiker	0 hard / -80 soft	M_An1I	120 min

Figure 4.2: Breakdown of Costs: Each exam is punished with the same penalty.

The second approach calculates the proportional cost for each exam. The updated breakdown of costs is shown in Figure 4.3. The second variant is a noticeable improvement. The list contains only the exams of the first week as the exams in the second and third weeks are fine. Furthermore, the exam “M_Bsys1” has the lowest score (-275 soft – highest penalty) as on the day before and on the day after another exam takes place another exam.

Betriebssysteme 1	0 hard / -275 soft	M_Bsys1	120 min
Computernetze 1	0 hard / -144 soft	M_CN1	210 min
Diskrete Mathematik für Informatik	0 hard / -137 soft	M_DMI	120 min
Datenbanksysteme 1	0 hard / -7 soft	M_Dbs1	120 min

Figure 4.3: Breakdown of Costs: Each exam is punished with the actual penalty.

Note, the above-described problem was not about instructing the solver to do the right thing but rather about giving the user the correct insights on the solution's quality. The breakdown of costs does not affect what solutions the solver finds in any way. The decision of which solution is better is solely based on the total score [19]. Moreover, the scores in Figure 4.2 and Figure 4.3 are the weighted costs. The score function used to determine the total score of a solution will be described in section 4.6.

It is worth looking at a crucial implementation detail for constraints based on the cost function: The result of the cost function is a floating-point number. The OptaPlanner documentation recommends avoiding floating-point numbers in the score calculation. Arithmetic with floating-point numbers can lead to incorrect decisions, especially in planning problems. Decimals numbers, such as Java `BigDecimal`, is one alternative. However, the OptaPlanner team has shown in experiments that using `BigDecimal` makes the score calculation up to five times slower [20]. Therefore, we decided to multiply each cost value by a factor of 1,000 and then cast it to an integer.

4.1.3 Manual Scheduling

User Stories: D.1.3, D.1.10, D.1.16

The user can manually schedule an exam by defining the date, time, and one or multiple rooms as shown in Figure 4.4. During the solving process, manually scheduled exams will not be moved. In other words, the solver does not change the provided scheduling information by the user.

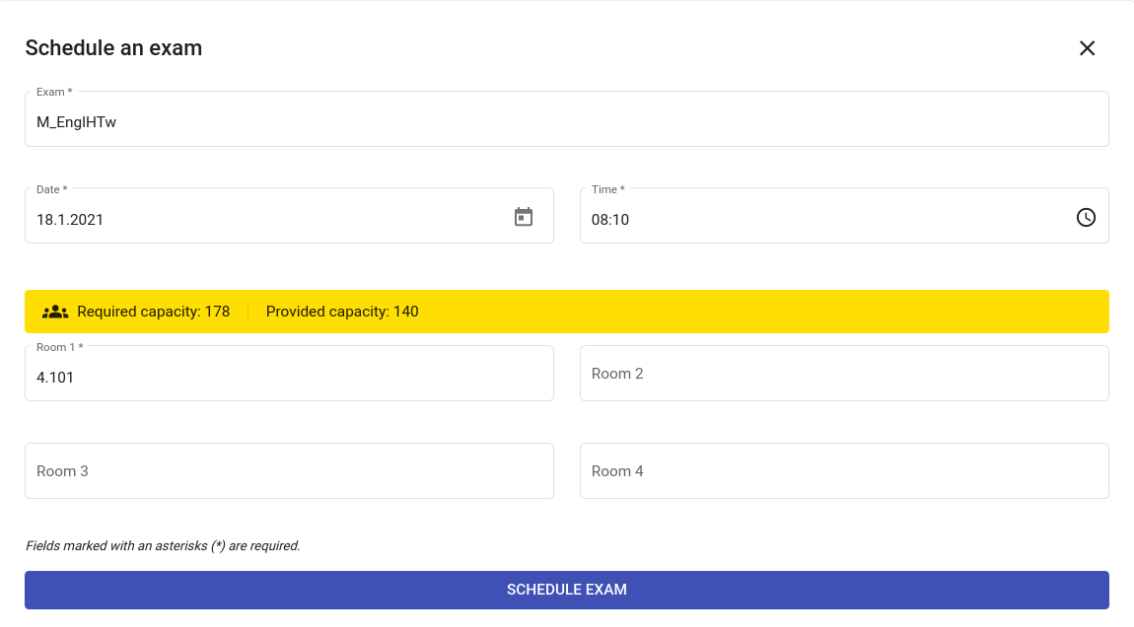


Figure 4.4: Dialog for Manually Scheduling an Exam: Despite the warning that the selected room has not enough capacity, the user can still schedule the exam.

Validation rules ensure that the user cannot schedule an exam outside the examination session or the daily examination time, and that at least one room is provided. Moreover, it is checked that the entered data is valid. Apart from these checks, no further validation rules are in place. This design decision allows the user to override the solver's choices and break any constraint if needed. Manually scheduling an exam is a powerful feature, and with great power comes great responsibility. To prevent an unintentional constraint violation, the system updates the solution's score on every change by the user . The user can then check for any constraint violations related to the manually scheduled exam and decide whether these are okay or not. Manual scheduling is always available, except while the examination schedule is being solved.

Originally, the plan was to implement partial locking feature, such that the user can only define the room or the date and time. Unfortunately, OptaPlanner does not yet support this feature and the possible workarounds are too complex [21]. Therefore, we had to put this feature on hold.

4.1.4 Unavailability Periods of Rooms

User Story: D.1.1

The user can define unavailability periods for rooms. A hard constraint ensures that no exams are scheduled in unavailable rooms. Modifying the unavailability periods is always possible, except while the examination schedule is being solved. Adding a new unavailability period triggers a solution's score update. Exams that are scheduled in an unavailable room will then be penalized.

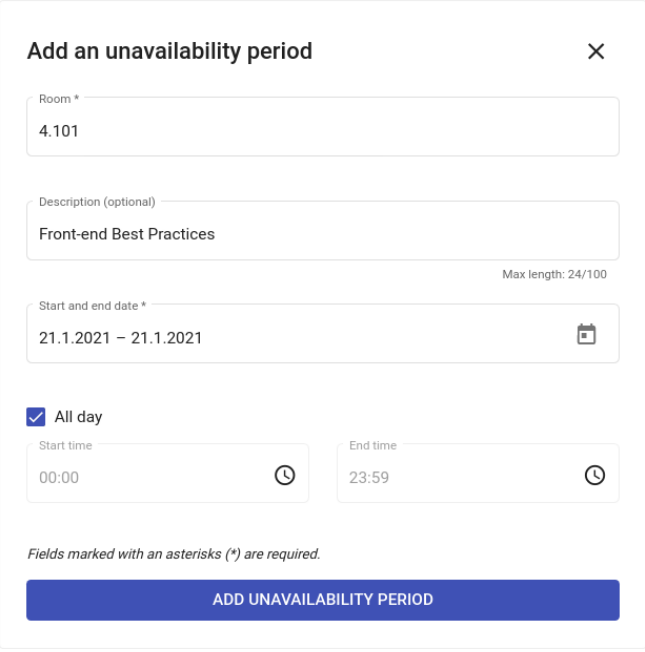


Figure 4.5: Dialog for Adding an Unavailability Period

4.1.5 Consideration of Room Types and Daily Examination Time

User Story: D.1.9

There are two types of exams:

- paper-based exams
- computer-based exams

A computer-based exam can only take place in a computer room, and a paper-based exam can only take place in a non-computer room. This requirement is a hard constraint, and one could implement another OptaPlanner hard constraint to ensure it. However, allowing only a subset of the available room is a relatively simple constraint. For such simple constraints, one can make use of dynamic value range providers. A value range provider is a property referencing a collection of the available values for a planning variable or a method that returns such a collection. The value range provider is annotated with `@ValueRangeProvider` and can be on the planning entity class, such that each planning entity has its own value range provider. With this, unsuitable rooms can be filtered out upfront. This way, the solver never assigns a value to a planning variable that would violate a hard constraint.

Dynamic value range providers can also be used to ensure that exams take place within the daily examination time. In this case, the value range provider for the starting time of an exam only contains the start times whose end time (start time + exam duration) does not exceed the daily examination end time.

4.1.6 Optional Constraint for Spring Semesters

User Story: D.1.24

The examination planning team differentiates between the fall and spring semesters when planning. In the spring semester, the examination session takes place in the summer. Therefore, the examination planning team plans exams in the afternoon only in rooms with air conditioning. Since this constraint depends on the type of semester, it must be enabled or disabled at run time. Figure 4.6 shows the option to enable or disable the constraint in the user interface. The configuration is stored for each schedule in the database.

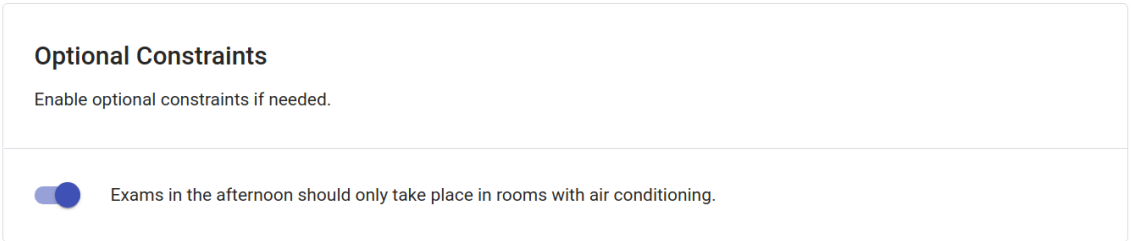


Figure 4.6: Slide Toggle for Optional Constraint

Optional constraints can be implemented with a dynamic constraint configuration. The Java annotation `@ConstraintConfiguration` defines a constraint configuration class that contains the base scores for one or multiple constraints. At run time, a configuration object is created based on the user settings, loaded from the database. This configuration object is then used in the score calculation. In order to disable a constraint, the base score is set to zero [22].

4.2 Runtime Performance Optimizations

Part of the challenge of this project was ensuring that the runtime performance of the resulting application is high enough so that it is usable in the everyday life of an exam scheduler, i.e., to plan the exams each semester.

We had several options to tackle this. The most effortless being to use the newest and most optimized runtime environments and to add the proper parameters. These optimizations can already improve the performance by a lot if the project was created a while ago and its versions are outdated. However, as this project is relatively new, the update did not bring any ground-breaking improvements. In our case, the points with the most promising improvement were optimizing the code itself and tweaking the solver configurations and its algorithms. A deep dive into those two approaches follows in the subsequent sections.

4.2.1 Runtime Profiling

Even with the most powerful machine and the most optimized runtime environment, if the code is inefficient, everything is inefficient. In our case, the application core is not the bottleneck, as it is only used to load the timetable from and into the solver. The critical part is the code that is executed by the solver. More specifically, the code that is executed by the part of the solver that checks all the constraints. To find such code, we used a profiler (VisualVM) to see which methods take how long to execute. Those methods can then be optimized as much as possible.

Initial Run

The initial run (Figure 4.7/Figure 4.8) was executed with the application, with no alterations of the code and no explicit focus on performance. It quickly became clear that the `TimeGrain` class caused a performance bottleneck. The self-time, the time spent in a method itself, accumulated over all methods in this class caused 27.9% of the total execution time.

A second bottleneck in the application was found in the exam class regarding the handling of rooms. 6.6% of the accumulated self-execution time and 3.1% of the accumulated total time, being the sum of the self-time and all self-times of the methods called by this method, was caused by checks regarding rooms.

Name	Self Time (CPU)		Total Time (CPU)	
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**startingMinuteOfDayToLocalTime** ()	105,813 ms	(11.5%)	146,036 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**<init>** ()	93,569 ms	(10.2%)	246,428 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**getTimeGrainAfter** ()	57,381 ms	(6.2%)	353,117 ms	(0.2%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	47,780 ms	(5.2%)	2,884,305 ms	(1.5%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**useSameRooms** ()	12,652 ms	(1.4%)	3,094,650 ms	(1.6%)

Figure 4.7: First Profiling – Top 5 – Sorted by Self-Time (Total Execution Time: 1,050 seconds = 17.5 minutes) – *An extended, unfiltered version can be found in appendix E.1*.

Name	Self Time (CPU)		Total Time (CPU)	
ch.ost.examscheduler.solvers.opta.domain.Exam.**useSameRooms** ()	12,652 ms	(1.4%)	3,094,650 ms	(1.6%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	47,780 ms	(5.2%)	2,884,305 ms	(1.5%)
ch.ost.examscheduler.solvers.opta.solver.RoomConstraints$$Lambda$1004.0x00000008408e8040.**test** ()	871 ms	(0.1%)	1,583,699 ms	(0.8%)
ch.ost.examscheduler.solvers.opta.solver.RoomConstraints$$Lambda$984.0x00000008408d9c40.**test** ()	1,903 ms	(0.2%)	1,513,725 ms	(0.8%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**haveCommonStudents** ()	1,152 ms	(0.1%)	1,387,676 ms	(0.7%)

Figure 4.8: First Profiling – Top 5 – Sorted by Total-Time (Total Execution Time: 1,050 seconds = 17.5 minutes) – *An extended, unfiltered version can be found in appendix E.1*.

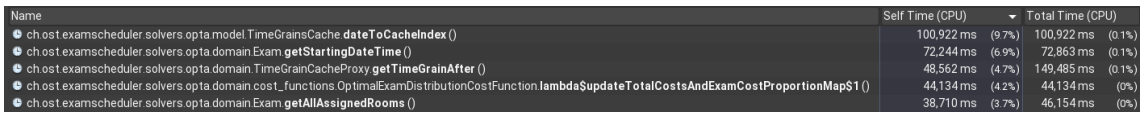
Tackling the Issues

Identifying the source of the bottleneck in the `TimeGrain` class was relatively straightforward. A `TimeGrain` was freshly created every time a new/different `TimeGrain` was required. This on-demand creation created an enormous overhead in object creation and garbage collecting unneeded/unused `TimeGrains`. A problem, easily solved by adding a cache that is prefilled before starting the solving process. One important point to mention: Adding a cache for classes used by the OptaPlanner must be done very carefully. OptaPlanner can run highly in parallel and clones its object instances to avoid causing issues across threads, entities, facts, and if configured, even nested properties of those. This cloning can cause tremendous troubles if the cached classes contain any state. Luckily, the `TimeGrain` is a value object and therefore wholly stateless.

Inefficient lookups and list operations mainly caused issues regarding the rooms. We could mitigate those by replacing the underlying data structure and reducing the number of operations it takes to generate the list.

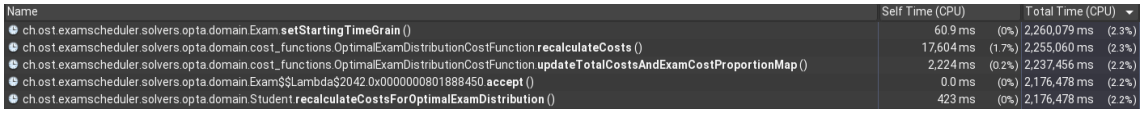
Optimized Version

Based on the above-described performance improvements and the new feature additions to the exam scheduler, a new performance profiling was made (Figure 4.9/figure 4.10).



Name	Self Time (CPU)		Total Time (CPU)	
ch.ost.examscheduler.solvers.opta.model.TimeGrainsCache.**dateToCacheIndex** ()	100,922 ms	(9.7%)	100,922 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getStartingDateTime** ()	72,244 ms	(6.9%)	72,863 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrainCacheProxy.**getTimeGrainAfter** ()	48,562 ms	(4.7%)	149,485 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**lambda$updateTotalCostsAndExamCostProportionMap$1** ()	44,134 ms	(4.2%)	44,134 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	38,710 ms	(3.7%)	46,154 ms	(0%)

Figure 4.9: Final Profiling - Top 5 - Sorted by Self Time (Total Execution Time: 1,050 seconds = 17.5 minutes) – *An extended, unfiltered version can be found in appendix E.2*.



Name	Self Time (CPU)		Total Time (CPU)	
ch.ost.examscheduler.solvers.opta.domain.Exam.**setStartingTimeGrain** ()	60.9 ms	(0%)	2,260,079 ms	(2.3%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**recalculateCosts** ()	17,604 ms	(1.7%)	2,255,060 ms	(2.3%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**updateTotalCostsAndExamCostProportionMap** ()	2,224 ms	(0.2%)	2,237,456 ms	(2.2%)
ch.ost.examscheduler.solvers.opta.domain.Exam$$Lambda$2042.0x0000000801888450.**accept** ()	0.0 ms	(0%)	2,176,478 ms	(2.2%)
ch.ost.examscheduler.solvers.opta.domain.Student.**recalculateCostsForOptimalExamDistribution** ()	423 ms	(0%)	2,176,478 ms	(2.2%)

Figure 4.10: Final Profiling - Top 5 - Sorted by Total Time (Total Execution Time: 1,050 seconds = 17.5 minutes) – *An extended, unfiltered version can be found in appendix E.2*.

We can quickly see that the self-time of the `TimeGrain` vanished completely, respectively moved to the `TimeGrainCache`. Furthermore, we could reduce the self-time of TimeGrain related operations from 27.9% down to 14.4%. Further reductions are hard to achieve without spending an enormous amount of time to enhance date and time calculations, which would most likely end up in an entirely custom-made date, time, and date-time class.

The room operations also lost some self-time, although not as significant as the time grain operations. The new data structure and the more performant list creation reduced the self-time from 50 seconds (17.5 minutes) to 45 seconds. Nothing significant on a small scale, but expanding this to a typical solving time of 24 hours, its impact should not be underrated.

When examining the results more carefully (especially the extended version in section E.2), a new player stands out – the `OptimalExamDistributionCostFunction`. This function is the new addition required by the constraints that a student and a regular semester should have their exams as evenly distributed as possible. As this is a very computational heavy process, not much can be done to improve its performance other than caching its results, which is already done.

Thanks to the above-described improvements, the solver consistently keeps a score-calculation speed above 1,000, which is recommended by the OptaPlanner team. The score-calculation speed is a measurement that the OptaPlanner uses to describe the performance of the constraints.

4.2.2 OptaPlanner Benchmarking

In section 3.3 we discussed the algorithms that are available to use and which features they have. We have also created a shortlist of algorithms that seem to be the most promis-

ing ones based on the research. Based on that, some test runs on our concrete problem have to be made to compare those different algorithms. The OptaPlanner provides a tool for that, which also generates a report in the end.

Configuration

As a starting point, we wanted to compare the different algorithms to the default algorithm used by the OptaPlanner. Figuring out what the default algorithm is, was more difficult than expected. As it turned out, the OptaPlanner team mentioned nowhere in their documentation which configurations are used by default. Therefore, our initial assumption was that it uses the most basic one, the hill-climbing algorithm. It quickly transpired that this was not the case, as running the benchmark with it was firstly tremendously slow and secondarily resulted in no reasonable solutions. Therefore, the OptaPlanner source code had to be consulted. In a deep-dive session, after figuring out the internals of the OptaPlanner, a no-configuration fallback was found. It was the Late-Acceptance-Hill-Climbing algorithm ([23]). This finding also explained the relatively good results we saw in our tests where we did not configure any algorithm.

With this knowledge, we assembled a benchmark suite that compares:

- the default algorithm LAHC as a basis
- a reference version of the default algorithm LAHC where all configuration parameters are defined manually
- a modified default algorithm LAHC version based on a remark by the OptaPlanner team
- the default configuration but with SCHC instead of LAHC, as listed in the shortlist (subsection 3.3.5)
- the SCHC version with the adjustment of the OptaPlanner team remark
- the simulated annealing algorithm, as listed in the shortlist (subsection 3.3.5)
- great deluge as a version that should perform slightly worse based on the research

The configuration file with all the details can be found in section F.2. Each test runs until no new solution was found for four hours.

Results

Solver	Total	Average	Standard Deviation	Problem
				Problem_0
Simulated Annealing 4 !	-1hard/-76082soft	-1hard/-76082soft	0.0E0/0.0E0	-1hard/-76082soft 4 !
GREAT_DELUGE 5 !	-45hard/-87649soft	-45hard/-87649soft	0.0E0/0.0E0	-45hard/-87649soft 5 !
LAHC (Values from Default) 6 !	-80hard/-80963soft	-80hard/-80963soft	0.0E0/0.0E0	-80hard/-80963soft 6 !
LAHC - Modified (Accepted Count Limit - 4) 1	0hard/-66086soft	0hard/-66086soft	0.0E0/0.0E0	0hard/-66086soft 1
SCHC - (Accepted Count Limit - 1) 0	0hard/-63843soft	0hard/-63843soft	0.0E0/0.0E0	0hard/-63843soft 0
SCHC - (Accepted Count Limit - 4) 3 !	-1hard/-61867soft	-1hard/-61867soft	0.0E0/0.0E0	-1hard/-61867soft 3 !
DEFAULT (LAHC) 2	0hard/-76924soft	0hard/-76924soft	0.0E0/0.0E0	0hard/-76924soft 2

Figure 4.11: Benchmark Results *(green highlighted: winning algorithm, orange (!): not 0hard, gray numbers: ranks)*

Looking at the results (Figure 4.11), we see that there is no huge difference between the algorithms regarding score. Surprisingly, our manual configured copy and the default configuration itself did not perform equally well. The OptaPlanner Team must have added some additional special configuration that could not be straightforwardly found in the source code. The Great-Deluge algorithm performed better than expected, but everything else pretty much went as we thought based on the research. The ”winner” is the Step-Counting-Hill-Climbing algorithm. This is the same result as the referenced papers suggested. Eventhough, the default algorithm which lands on place two stopped much faster (see Figure 4.12 - 5th (pink) vs last (salmon) bar) with a similar result, the OptaPlanner chooses the score over the solving time when ranking the algorithms. If we check the solving speed of the algorithms (Figure 4.13), we see that both algorithms behaved very similarly, the SCHC version just found some extra solutions that reset the stopping time.

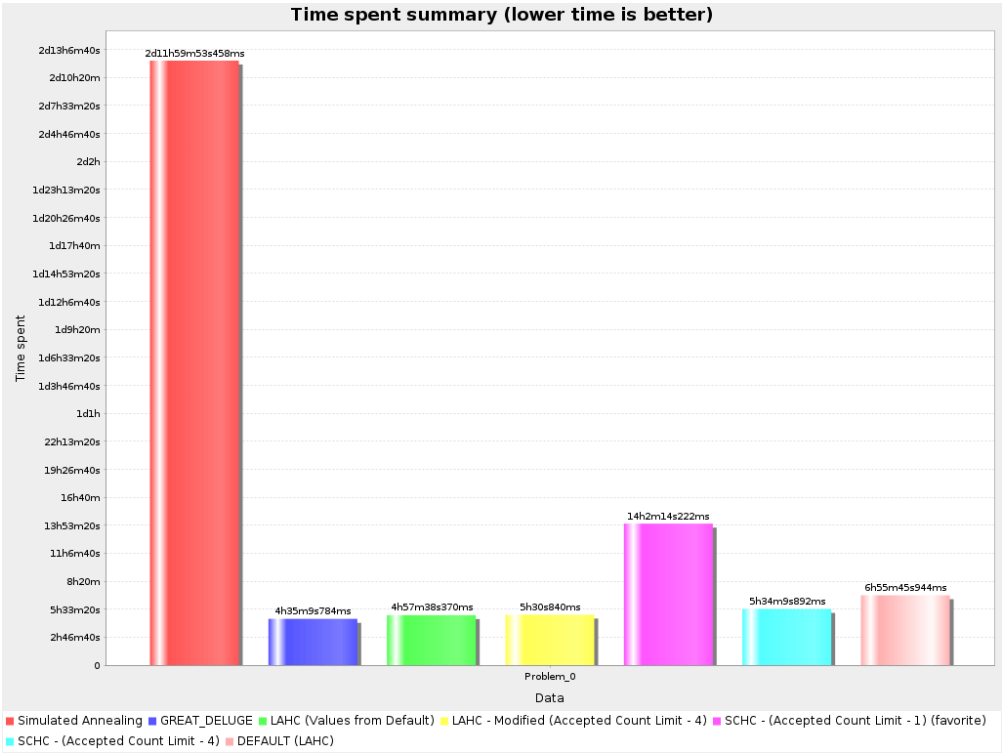


Figure 4.12: Benchmark Results – Solving Time (Runs stop after no new solution was found for four hours)

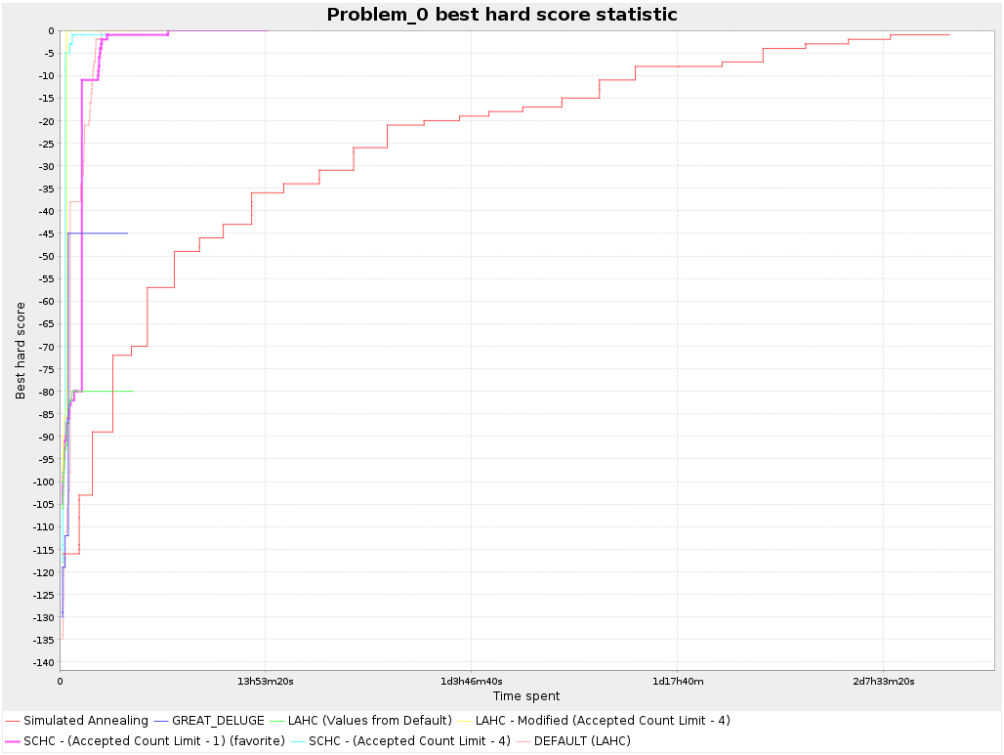


Figure 4.13: Benchmark Results – Algorithm Speed

4.2.3 OptaPlanner Configuration and Algorithm Choice

Based on those results, we select the SCHC algorithm as the best option for the production version of the exam scheduler (Listing 4.2). Additional tweaking could bring some small improvements, but the return on invest ratio is too small to further focus on that.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver">
    <moveThreadCount>AUTO</moveThreadCount>
    <solutionClass>ch.ost.examscheduler.solvers.opta.domain.ExamTimetable</ ↵
solutionClass>
    <entityClass>ch.ost.examscheduler.solvers.opta.domain.Exam</entityClass ↵
>
    <scoreDirectorFactory>
        <constraintProviderClass>ch.ost.examscheduler.solvers.opta.solver. ↵
constraints.TimeTableConstraintProvider
        </constraintProviderClass>
        <constraintStreamImplType>DROOLS</constraintStreamImplType>
        <initializingScoreTrend>ONLY_DOWN/ONLY_DOWN</initializingScoreTrend ↵
>
    </scoreDirectorFactory>
    <constructionHeuristic>
        <constructionHeuristicType>FIRST_FIT_DECREASING</ ↵
constructionHeuristicType>
    </constructionHeuristic>
    <localSearch>
        <unionMoveSelector>
            <cacheType>PHASE</cacheType>
            <changeMoveSelector>
                <filterClass>ch.ost.examscheduler.solvers.opta.solver. ↵
filters.AllRoomsUnassignedChangeMoveFilter
                </filterClass>
            </changeMoveSelector>
        </unionMoveSelector>
        <acceptor>
            <stepCountingHillClimbingSize>400</stepCountingHillClimbingSize ↵
>
        </acceptor>
        <forager>
            <acceptedCountLimit>1</acceptedCountLimit>
        </forager>
    </localSearch>
</solver>
```

Listing 4.2: Exam Scheduler Solver Configuration

4.3 Examination Schedule Visualization with Live Updates

User Story: D.1.26

The visualization of examination schedules (Figure 4.14) was not only re-styled but also received some additional features and performance optimization. However, the initial loading of the exam timetable caused performance problems. An examination schedule contains about 200 exams. Rendering all these items made the application unresponsive for a noticeable time. To solve this issue, we outsourced the rendering of the visualization into a separate background thread with the help of a web worker.

A significant improvement is live updates, enabled by a WebSocket between the client and the server. If the solver finds a new solution, the scheduling information of the exams and the updated score information are pushed to the client. The client then automatically updates the examination schedule and the solution's score. With this, the user does not have to refresh the page to get the latest state during an active solving process.

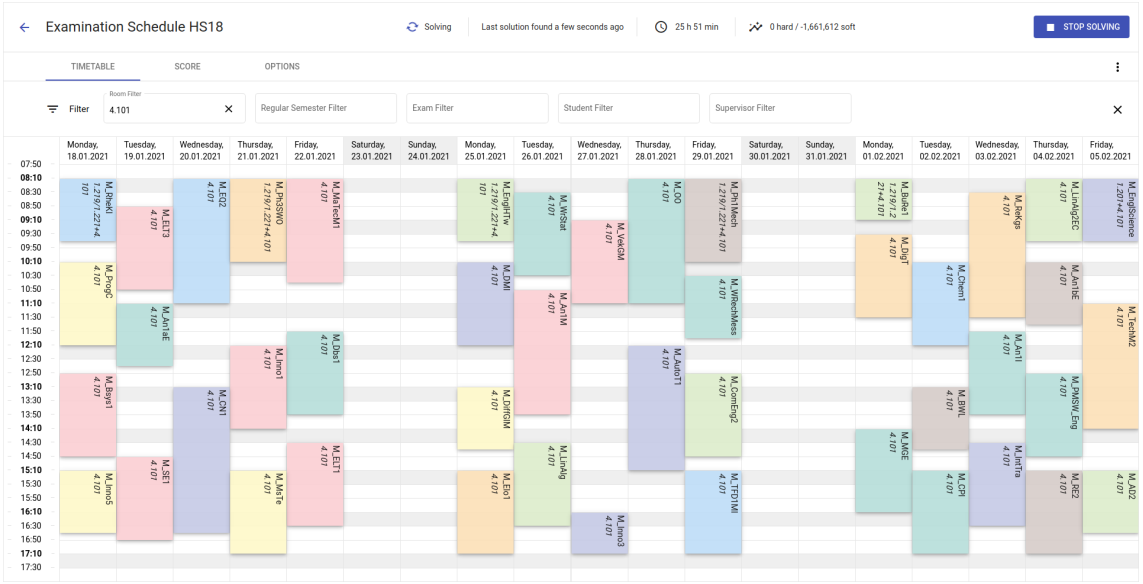


Figure 4.14: Exam Scheduler User Interface: The visualization shows only the exams scheduled in the school hall (room 4.101) as a corresponding room filter is activated.

4.4 Data Import/Export and Integration

User Stories: D.1.4, D.1.5, D.1.13, D.1.18, D.1.19

An interactive assistant (Figure 4.15) helps the user to create new examination schedules. In three simple steps, the user is asked to provide all the information needed for a new examination schedule.

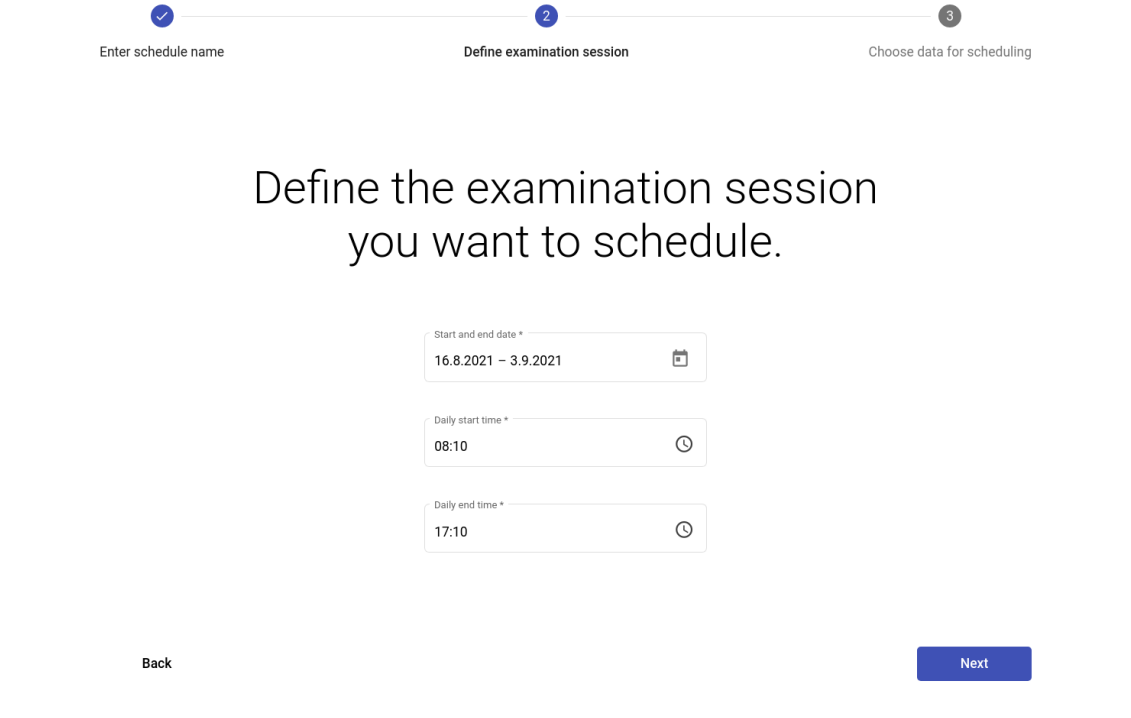


Figure 4.15: Step Two of the Creation Wizard: The user is asked to define the examination session and the daily examination time.

The data to be scheduled can be provided via the Excel files the examination planning team is familiar with. Importing the data with Excel files is error-prone. Although, the existing import API detects most validation violations, the existing validation rules only check the validity of a single row. In order to ensure all imported data is valid, we added additional validation rules that check the validity across multiple rows and even different files. For instance, a validation rule is in place so that all exam identifiers must be unique. With this, the user cannot accidentally provide two exams with the same id. Finding invalid data in an Excel file is a bit cumbersome. It is like finding the needle in the haystack. To overcome this problem, we point out the exact cell that is causing the problem, as shown in Figure 4.16.

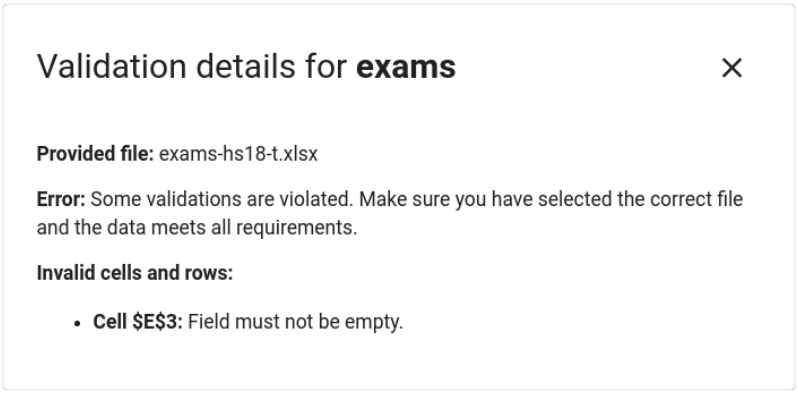


Figure 4.16: Validation Details for Exams: The user can see the cells causing the problem.

A newly created examination schedule does not contain any exams yet. The user can either manually schedule exams or start the solving process to plan all exams. Once a plan becomes final, the user can lock it. Locked examination schedules are read-only to avoid any unwanted changes. The user can export examination schedules as a spreadsheet or a JSON file. The spreadsheet contains the schedule information in a human-readable form, whereas the data in the JSON file is compatible with the administration software of the university. So the generated examination schedules can be seamlessly processed further. For example, in order for the students to see their examination plan in the university's administration tool.

4.5 Quality Assurance

To ensure the quality of our application, we have written around 840 tests for all classes containing some logic. Integration tests are in place for the database queries, data model, the Excel importer, WebSocket logic, and parts of the REST API. We have not set an explicit test coverage goal, as we strive for a close to 95% coverage of the solver logic and a coverage as high as possible for the rest of the application. The solver currently has a line test coverage of 94% and an overall line test coverage of 77%. Solver tests contain tests regarding the individual entities and their functionality and exhaustive tests regarding the individual constraints.

For the frontend, we have decided to do no UI tests, as the UI is not mission-critical, and only test classes containing some core logic like Angular pipes. This restriction was made due to the nature of UI tests being very time-consuming and having to be updated frequently.

Detailed code-, test-, git- and CI/CD stats can be found in Appendix B

4.6 Score Function

This section describes the score function used by the exam scheduler to quantify the quality of a specific examination schedule. The higher the score, the better the solution. The goal is to maximize the score function. Table 4.1 lists all constraints that we implemented with the OptaPlanner. These constraints are the basis of the score function.

Table 4.1: All Implemented OptaPlanner Constraints of the Exam Scheduler with its Initial Penalties (negative base scores) and Multipliers

ID	Constraints	Penalty[1]	Multiplier
1	**Room related constraints**		
1.1	Room has two exams at the same time	1,000 hard	1 *(none)*
1.2	Room has no break between exams	1,000 hard	1 *(none)*
1.3	Exam is scheduled in unavailable room	1,000 hard	1 *(none)*
1.4	Room has not enough space	800 hard	1 *(none)*
1.5	Exam has no room assigned[2]	2,000 hard	1 *(none)*
1.6	Exam unnecessarily occupies rooms[2]	200 hard	1 *(none)*
2	**Student related constraints**		
2.1	Students have two exams at the same time	20 hard	# common students
2.2	Students have not enough break time between exams on the same day	20 hard	# common students
2.3	Students have too many exams on the same day	10 hard	# students having too many exams
2.4	Students have two exams on the same day	1,000 soft	# students with 2 exams
3	**Examiner related constraints**		
3.1	Examiner has two exams at the same time	800 hard	1 *(none)*
4	**Time related constraints**		
4.1	Exam overlaps with lunch time	7,500 soft	if fully: 2, else 1
5	**Optimal exam distribution related constraints**		
5.1	Exams of a regular semester are not optimally distributed	500 soft	Cost for optimal exam distribution
5.2	Exams of a student are not optimally distributed	10 soft	Cost for optimal exam distribution

[1] In the score calculation penalties are treated as negative scores.
[2] These constraints do not ensure business constraints, but are necessary due to the way we implemented the support for scheduling an exam in multiple rooms.

Colors: Hard constraints Soft constraints

Note that not all constraints listed in section 2.3 are ensured with the help of OptaPlanner constraints. For instance, the constraint that computer-based exams can only take place in computer rooms and paper-based exams cannot take place in computer rooms does not need an OptaPlanner constraint. Instead, we filter the available rooms for each exam upfront. This way, an exam can only be scheduled in an appropriate room.

The OptaPlanner supports negative scores (penalties) and positive scores (rewards). Both scoring techniques are based on constraints. When a constraint is activated as a negative constraint is violated, or a positive constraint is fulfilled, it is called a *constraint match* [24].

Having only negative constraints (Table 4.1) comes with the advantage of a well-known maximum of `0`. A score consists of a "hard" and "soft" value because we implemented a two level score model. " Hard constraints are weighted against other hard constraints. Soft constraints are weighted against other soft constraints. Hard constraints always outweigh soft constraints, regardless of their respective weights." [25]

The total score of all hard constraints respective the total score of all soft constraints is calculated according to Definition 5.

Definition 5. *Let C be a set of constraints, s_c the constraint base score, M_c the matches of the constraint $c \in C$, and w_m the weight of the match $m \in M_c$. The total score S for the given constraints C is calculated as follows:*

$$S = \sum_{c \in C} \sum_{m \in M_c} w_m \cdot s_c \tag{4.1}$$

As an example, an examination schedule could have a score of `0hard/-7500soft`. The value `0hard` indicates that no hard constraints are violated. Therefore, the score belongs to a feasible examination schedule. Given the penalties in Table 4.1, we can further say that the solution with a score of `-82soft` is quite good. However, the second interpretation is vague as the input data is not specified in this example. To interpret the score of a solution in a meaningful way, it must always be put into perspective with the original problem.

The cost calculation is part of the constraint definition and is implemented with the OptaPlanner's ConstraintStream API. Listing 4.3 shows the implementation for the constraint "Students have two exams at the same time". For this specific constraint, a constraint match is associated with a pair of exams. The filters on line `9` and `12` only let exam pairs through that violate the constraint. The remaining exam pairs are the constraint matches and are penalized. The method call `.penalize()` on line `13` conceptually is the multiplication of the constraint base score s_c with the weight of the match w_m from the mathematical formula for the score calculation (Definition 5).

The static field `HardMulti.STUDENT_HAVE_TWO_EXAMS_AT_THE_SAME_TIME` on line 15 defines the constraint base score. The constraint match weight is calculated with the function `Exam::numberOfCommonStudents` on line 17. We refer to the functions that calculate the constraint match weights as the *constraint multipliers*.

```
1  public class StudentConstraints extends AbstractConstraints {
2    // ...
3    Constraint studentsHaveTwoExamsAtTheSameTime() {
4    return constraintFactory
5      // Two exams get grouped to a pair.
6      .fromUniquePair(Exam.class)
7      // This filter only lets through exam pairs
8      // whose exams overlap in time.
9      .filter(Exam::areOverlapping)
10     // This filter only lets through exam pairs
11     // whose exams have common students.
12     .filter(Exam::haveCommonStudents)
13     .penalize("Students have two exams at the same time",
14        // Constraint base score
15        HardMulti.STUDENT_HAVE_TWO_EXAMS_AT_THE_SAME_TIME
16        // Constraint multiplier
17        Exam::numberOfCommonStudents);
18   }
19 }
```

Listing 4.3: Implementation of the Constraint “Students have two exams at the same time”

4.6.1 Constraint Base Score

The constraint base score s_c in Definition 5 is the parameter to weight the constraints against each other. Finding optimal weighting parameters is an experimental process. We started with an initial guess. We weighted the hard constraints that are physically impossible or do not have an easy workaround slightly higher than the others. For the soft constraints, we tried to chose the weighting parameters such that the priority of the constraints according to section 2.3 is ensured.

Defining the base score for constraints with a multiplier is particularly tricky because the constraint score depends on the multiplier which in turn depends on the input data. If the exam scheduler is used with a much larger dataset adjustments on the weighting parameters will probably be necessary. Within this project’s scope, the exam scheduler is only intended to be used for the OST campus in Rapperswil-Jona. Therefore, we will not further investigate this issue.

To come up with an initial guess for the base scores of constraints having multipliers, we considered the maximal possible score of each constraint. Table 4.2 shows the underly-

ing calculation. We choose the base score of each constraint in a way that the resulting product of the base score and the scaling factor is the same for all constraints. For the constraints regarding an optimal distribution, the cost function is also included in the product. With this approach, we wanted to avoid that the maximum score of one constraint is far off compared to the other constraints, which would then always outweigh the constraints having much lower scores. As a next step, we wanted to fine tune the weighting. However, since the test results were already satisfying with the initial setting, further adjustments of the weights did not seem to be needed.

Table 4.2: Reasoning for Base Scores of Constraint having Multipliers
(Base scores of all constraints can be found in the column “Penalty” of Table 4.1.)

Constraint	Base score	Scaling Factor	Cost Function Worst Case		Product
Exam during lunch	7,500	200	Exams		1,500,000
Two exams on same day	10,000	1,500	Students		1,500,000
Distribution regular semesters	500	30	Regular semesters	100	1,500,000
Distribution students	10	1,500	Students	100	1,500,000

4.6.2 Constraint Multiplier

Using constraint multipliers has the advantage that we can define the score improvement on a much more granular level. In other words, thanks to constraint multipliers the OptaPlanner knows where it gets the most bang for its buck [26]. This is especially useful for soft constraints as these constraints should be fulfilled but might be broken if necessary.

The effect is best illustrated with an example. Figure 4.17 shows an examination schedule with four exams. The exam pairs (Exam A, Exam B) and (Exam C, Exam D) fulfill the following conditions:

- Both exams of the pair take place on the same day.
- There is at least one student who writes both exams of the pair.

Therefore, each exam pair is a constraint match for the constraint “Students have more than one exam on the same day”. Without a constraint multiplier, both constraint violations would receive the same penalty. However, the exam pair (Exam C, Exam D) has 40 times more students. So, re-scheduling Exam C or Exam D instead of Exam A or Exam B could increase the schedule quality for more students and should therefore be preferred. Thanks to the constraint multiplier, this intent is ensured because the re-scheduling of Exam C or Exam D instead of Exam A or Exam B improves the score significantly more.

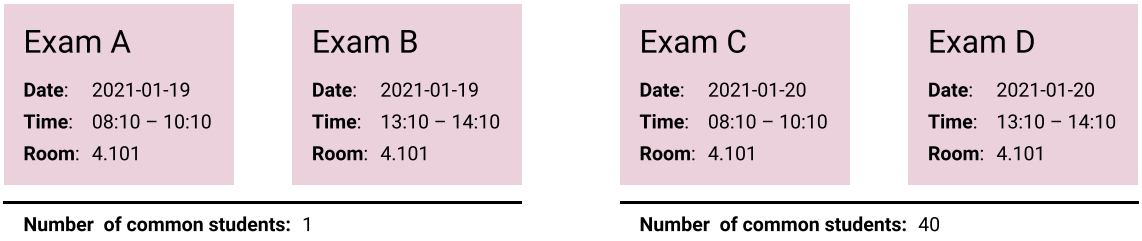


Figure 4.17: Sample Examination Schedule with Four Exams

4.6.3 Visualization of Solution Score

User Story: D.1.25

The solver uses the total score to decide whether a new solution is better than the current best solution. However, a score like `-13,020hard/-314,181soft` is a black box for the user. Without further insights, the user can barley say anything about the solution, except that it is feasible or not. Therefore, the user interface provides two ways the user can use to explorer the score:

- The constraint details (Figure 4.18) break down the score for each constraint and list all exams that violates the constraint.
- The indictment details (Figure 4.19) point out all violated constraints for each exam.

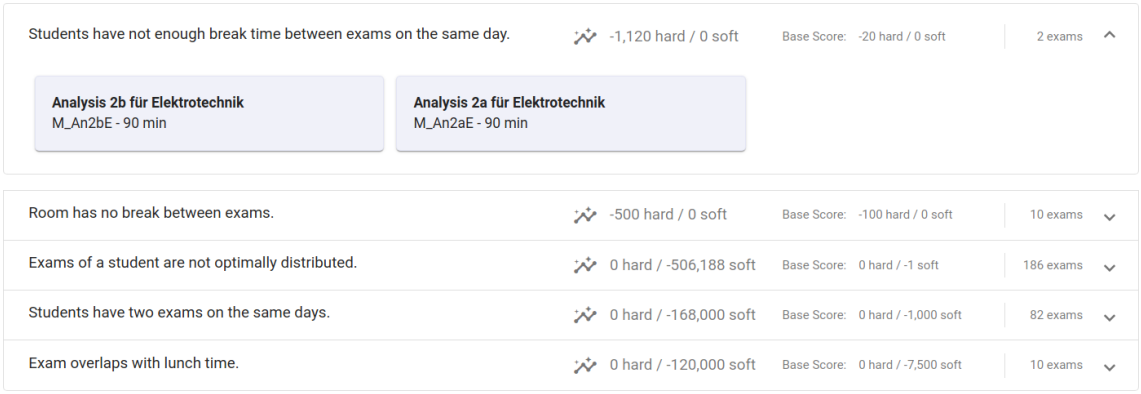


Figure 4.18: Constraint Details: Two exams violate the constraint “Students have not enough break time between exams on the same days”.

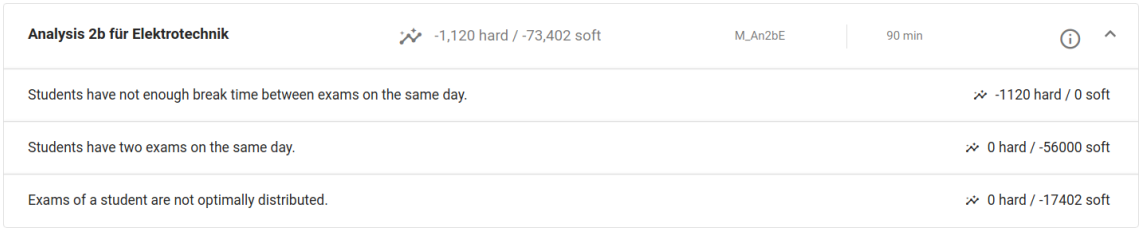


Figure 4.19: Indictment Details: The exam “Analysis 2b für Elektrotechnik” violates one hard constraint and two soft constraints.

Chapter 5

Results

5.1 Test Setup

For testing the exam scheduler, we had to set up a test environment. For deploying a version directly from the GitLab CI/CD pipeline, we used a Kubernetes cluster running on the Azure cloud. The underlying server had 16 vCPUs @ 3.4 GHz running on an Intel Xeon Platinum 8272CL and 32 GB of memory. All the tests running on this machine led to a total cost of CHF 300, which the IT department of the OST covered.

For running manual tests and the benchmarks, a second server was used. This server had 64 vCPUs @ 1.9 GHz running on an Intel Xeon X7550 and 64 GB of memory. This server is located on-premise and did not cause any costs for us.

All comparisons shown in this chapter were always executed in the same environment to not bias the results in any way. The presented results are based on the test sets described in Table 5.1.

Table 5.1: Selected Test Sets for the Exam Scheduler

Test Set	Exams	Registrations	Students	Rooms[1]	Session[2]	Time
HS18	189	9,005	1,357	20	3 weeks (15 d)	08:10 – 17:10
FS21	187	7,791	1,191	20	3 weeks (15 d)	08:10 – 17:10

[1] All rooms are always available throughout the whole examination session.
[2] At the weekends no exams are scheduled.

5.2 Key Performance Indicators (KPIs)

The constraint scores as shown in Figure 4.19 can be used to compare different solutions, but they are kind of an abstract measure for the quality. Moreover, an actual examination schedule has some exceptions that the problem model does not fully cover. For instance, two exams must take place in the same room at the same time. These exceptional exams must be manually scheduled, resulting in some constraint violations. Although, in this case, the violations are wanted, they still result in a penalty. Because of these two reasons, the constraint scores are not the best measure to compare the quality of an examination schedule generated by the exam scheduler with the one created by a human. To overcome this issue, we want to define three KPIs:

1. Histogram of all student's optimal exam distribution costs and the corresponding cumulative frequency curve
2. The number of exams that fall into the lunch break (12:30 to 12:50)
3. The number of students that must write two exams on the same day.

The listed KPIs cover all soft constraints. The hard score is expected to be zero as the examination schedule must be feasible. If there are any violations of hard constraints, exceptions must justify them. Otherwise, the severity is pointed out.

5.3 Optimization for Regular Semesters is Counterproductive

Optimizing the exam distribution within a regular semester is supposed to optimize the examination schedules of as many students as possible as within the regular semester, most students take the exam. However, while working with the exam scheduler and analyzing the input data, we noticed that the allocation of a regular semester is sometimes incorrect. Assigning a module/exam to a regular semester is not an easy task because of the free module choice at the OST.

In traditional exam scheduling, the regular semesters help the (human) exam scheduler to distribute the exams evenly. It is the only instrument to achieve balanced examination schedules for students. However, the exam scheduler software has an additional constraint that ensures an optimal exam distribution for students. Therefore, the question arises whether the optimization for regular semesters does really make a difference. To answer this question, we carried out two test runs. In the first run, the optimization for regular semesters was active, and in the second one, we disabled it.

Figure 5.1 shows the histogram of the optimal exam distribution costs of all students. The corresponding cumulative frequency curve is shown in Figure 5.2. The comparison of the two test runs states that the optimization for regular semesters has a negative impact on the optimal exam distribution for the students.

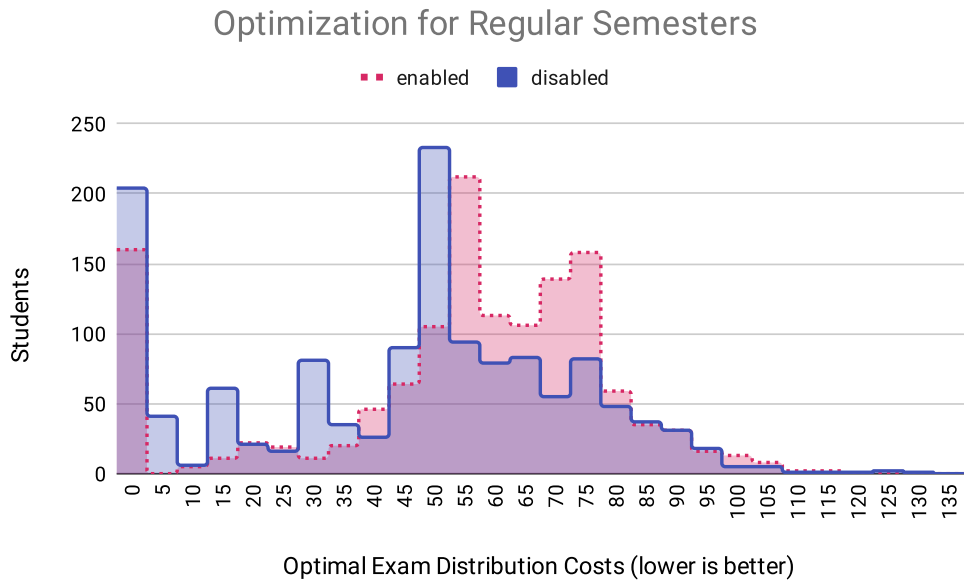


Figure 5.1: Histogram of Students' Optimal Exam Distribution Costs (Test Set HS18)

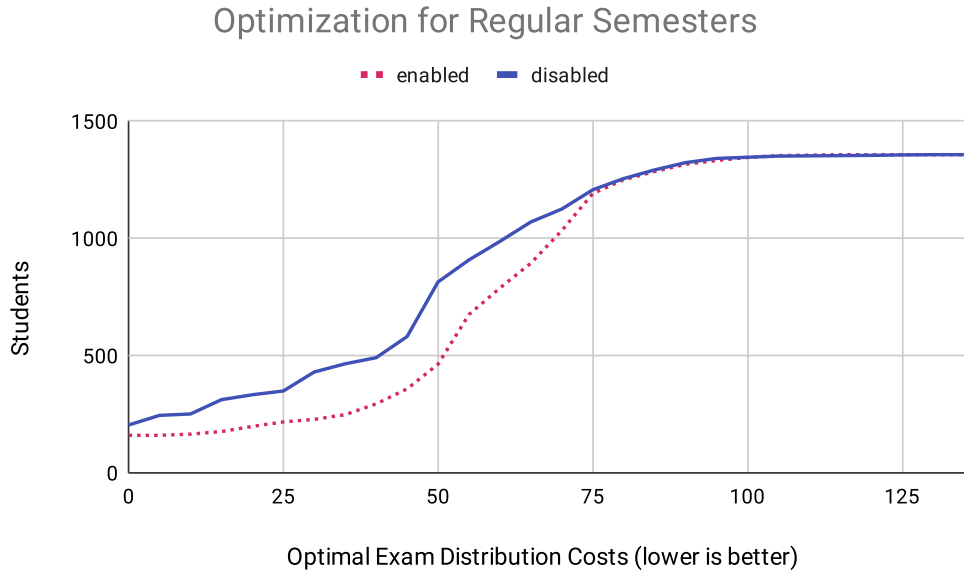


Figure 5.2: Cumulative Frequency Curve of Students' Optimal Exam Distribution Costs (Test Set HS18)

The constraints regarding an optimal exam distribution for students and regular semesters conflict with each other, resulting in worse examination schedules. Therefore, we entirely disable the optimization regarding regular semesters. Removing the constraint from the score calculation also brings a performance win as the computation for it is quite computationally expensive.

5.4 Comparison with an Actual Examination Schedule

For the software to be used productively, it must generate examination schedules that are at least as good as those created by a human planner. Therefore, we compare the actual examination schedule “FS21” with a solution generated by the examination scheduler. For the comparison, we created a new exam timetable and manually scheduled all exams according to the Excel file containing the actual, human-made examination schedule.

5.4.1 Creating a Solution with the Exam Scheduler Software

To generate the FS2021 examination schedule, we could not simply import the data and start the solving process as the actual schedule contains some exceptions that violate constraints. Hence, we had to schedule these exams manually. Figure 5.3 shows the initialized exam timetable with 31 exceptional exams. Exams that run throughout the day are reserved days for specific oral exams.

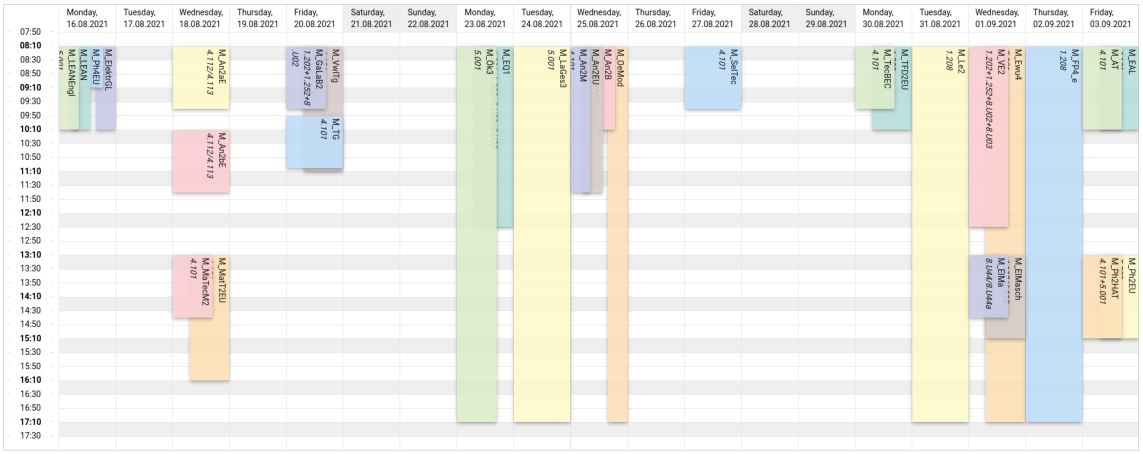


Figure 5.3: Initialized Examination Schedule “FS21”: 31 of 187 exams require manual scheduling due to exceptions that violate constraints.

The manual initialization phase took us about two hours. Once all exceptional exams were scheduled, we could start the automated solving process. After about 13 minutes, the solver finished the construction phase, i.e., all exams are scheduled. Surprisingly, the

found solution has no lower/worse hard score than the initialized schedule and is therefore a feasible solution. This is probably related to the fact that the most difficult plannable exams are already scheduled manually. Moreover, we enhance the construction heuristics by defining a planning difficulty that considers the number of students and exam duration. The most difficult exams are scheduled first.

After the construction phase, the local search phase started. Figure 5.4 shows the soft score development during the local search phase. In the first 45 min, the solver made the most significant improvement. Then for about three hours, the improvements were minimal. However, after that period, the score improved quite a bit again. We stopped the solving process after about ten hours as the solver had not found a better solution for three hours.

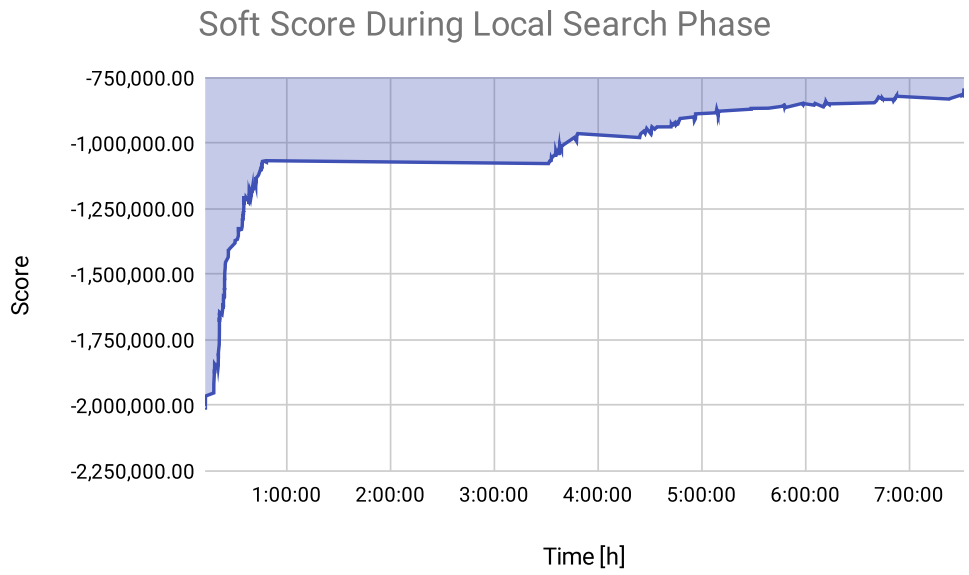


Figure 5.4: Soft Score Development During the Local Search Phase of the Test Set FS21 on the Azure Cloud

5.4.2 Generated versus Actual Schedule

Examining the actual examination schedule reveals that the constraint “Students have too many exams on the same day.” is violated. The hard score for this constraint is -330. Considering the base score of this constraint, one can say that it happens 33 times that a student must write two exams on the same day, where one exam lasts longer than 120 minutes. Moreover, the minimum break time of two hours between exams for students is not always guaranteed. According to the defined constraints, the actual examination schedule is not a feasible solution. In contrast, the generated examination schedule is a feasible solution as it does not violate any hard constraint.

We noticed that the hard constraint “Examiner has two exams at the same time.” is often violated. As these violations are not exclusively due to exceptionally scheduled exams, it should be considered to downgrade the constraint to a soft constraint.

Table 5.2 compares the soft constraints details of the actual and the generated exam timetable. The generated examination schedule has a higher total score and performs better regarding all soft constraints. Interestingly, there is not much of a difference in the optimal exam distribution. However, in the other constraint, the generated examination schedule has a significantly higher score.

Table 5.2: Actual vs. Generated Schedule for FS21 – Soft Score Constraint Details

Constraint	Soft Score	
	Actual Schedule	Generated Schedule
Exams of a student are not optimally distributed.	-520,234	-506,188
Exam overlaps with lunch time.	-442,500	-168,000
Student have two exams on the same day.	-246,000	-120,000
Total	**-1,208,734**	**-794,188**

The histogram of all students’ optimal exam distribution costs (Figure 5.1) and the corresponding cumulative frequency curve (Figure 5.2) reveal that in both exam timetables, the optimal exam distribution is similar. However, in the generated solution, significantly more students have a schedule with zero costs. The costs for an optimal exam distribution are, in general, slightly higher than the costs seen during the verification of the cost function in section 3.1. The costs were lower because the examination schedules used to verify the cost function do not contain days with two exams, whereas in a real data set it happens that student must write two exams on the same day.

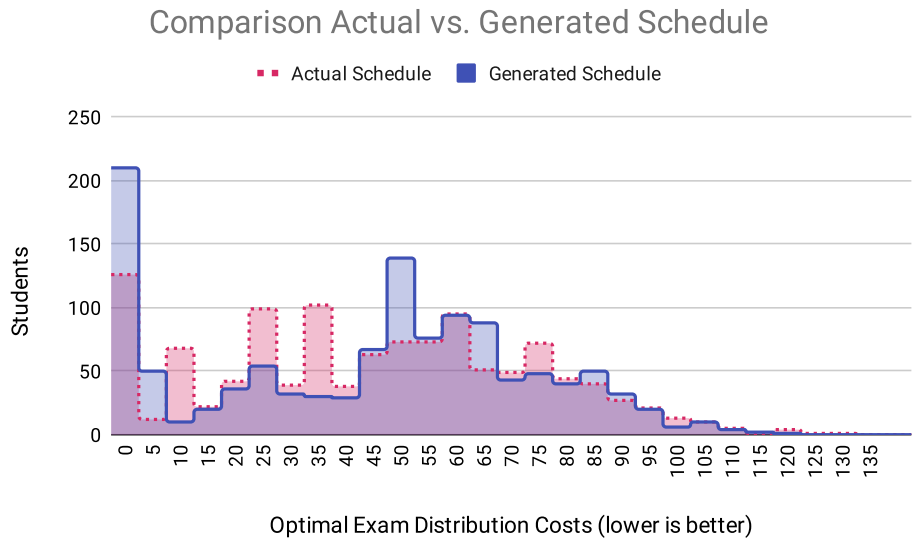


Figure 5.5: Actual vs. Generated Schedule for FS21: Histogram of Students' Optimal Exam Distribution Costs

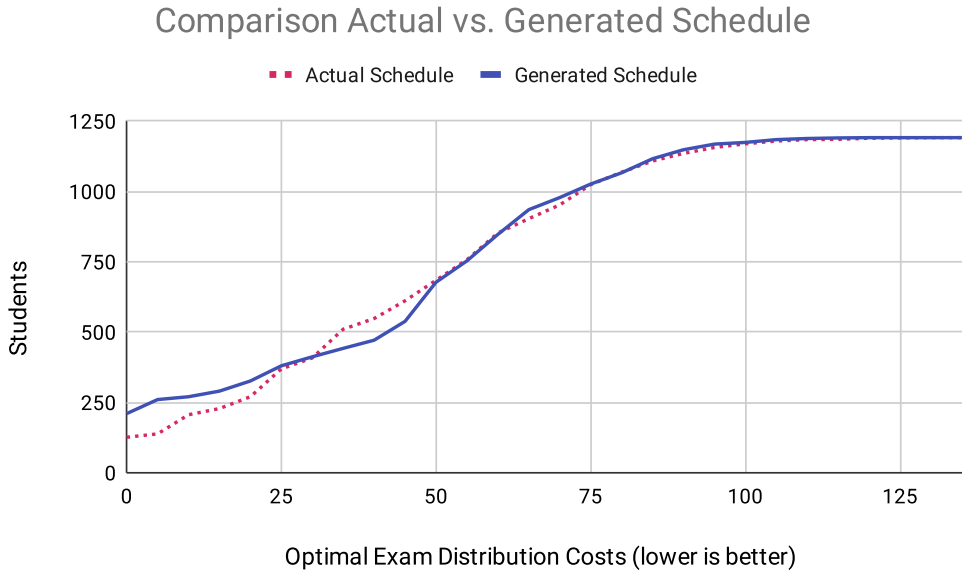


Figure 5.6: Actual vs. Generated Schedule for FS21: Cumulative Frequency Curve of Students' Optimal Exam Distribution Costs

Regarding students writing two exams on the same day, Table 5.3 contains precisely this information. In the actual examination schedule, 215 students have to write two exams on the same day. Whereas, in the generated schedule, the number could be significantly reduced down to 149.

Table 5.3: Actual vs. Generated Schedule for FS21 – Number of Students with Two Exams on the Same Day

Number of Days with Tow Exams	Number of Students with Two Exams on Same Day	
	Actual Schedule	Generated Schedule
1	186	131
2	27	17
3	2	1
Total	**215**	**149**

Finally, we count how many exams overlap with the lunch break. In the actual examination schedule, 12 of 187 exams entirely overlap with the lunchtime. In contrast, the generated schedule has no exams that take place during the lunch break. When comparing the visualization of both schedules (Figure 5.7 and Figure 5.8), one can clearly see that the actual schedule also has more exams that start at 12:30 or start at 12:50. Note again, the exams that last the whole day are blocked days for oral exams.

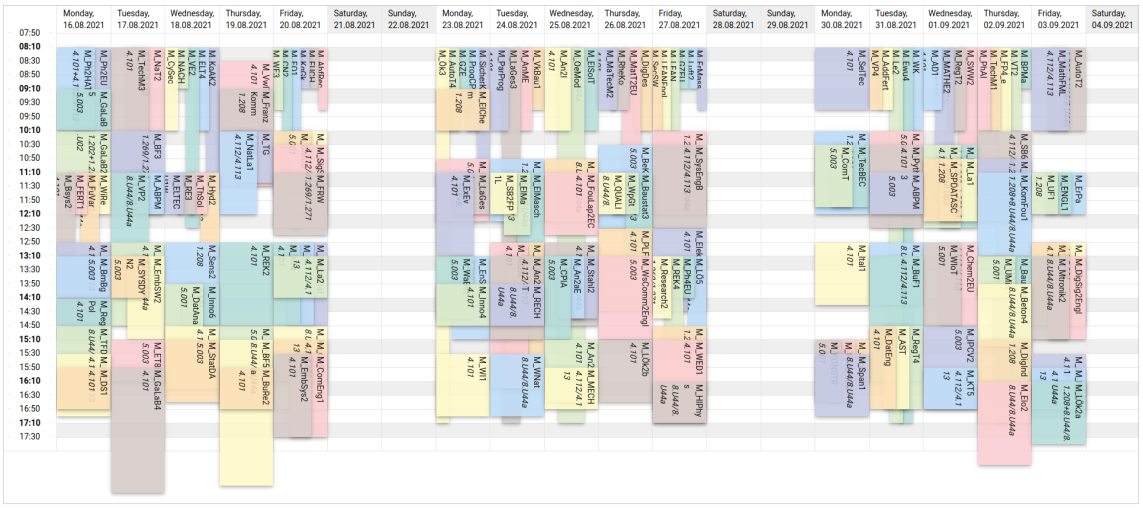


Figure 5.7: Visualization of the Actual Examination Schedule FS21

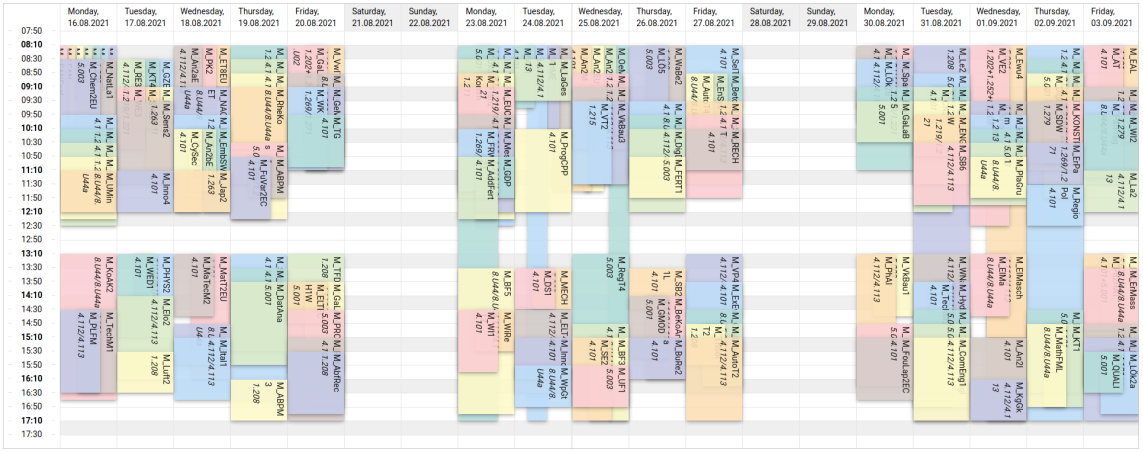


Figure 5.8: Visualization of the Generated Examination Schedule FS21

5.5 Comparison of Test Machines

Per default, the OptaPlanner runs in the reproducible mode. In this mode, two runs will execute the same code in the same order [27]. Even multithreaded solving is reproducible as long as the `moveThreadCount` is stable [28]. In order to max out the performance, we enabled auto-detection for the possible move threads. Enabling this option results in different solving processes across machines with different numbers of CPUs.

Figure 5.9 shows the comparison of the soft score development between the server on the Azure cloud and the server at OST for the test set “FS21”. The solving process looks quite different. The server on the Azure cloud is faster than the server at OST. However, if the server at OST runs long enough, it will eventually find a similarly good solution. After approximately 31 hours the server at OST found even a better solution than the solution found by the server on the Azure cloud. If we had run the solving process on the Azure cloud longer, the solver would probably have found a better solution as well.

Noticeable are the ups and downs of the score at the beginning of the test run on the server at OST. This behavior is due to the fact the the first solutions were not feasible. A solution with a higher hard score is always better, even if the soft score is drastically worse than the soft score from the previous best solution.

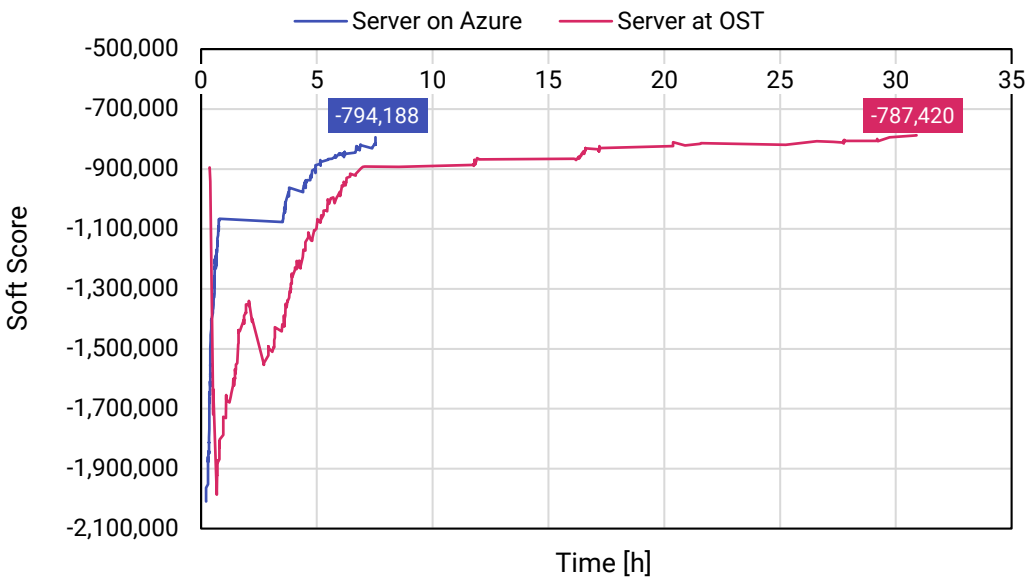


Figure 5.9: Soft Score Development During the Local Search Phase of the Test Set FS21 on the Azure Cloud and the Server at OST

5.6 Scalability

When discussing results, one may wonder how our solution scales regarding schedules with much more students and exams. To answer this, we conducted a test run with an input dataset of the size of the ETH Zurich:

- 15,000 students
- 700 exams
- 50'000 exam events / exam registrations
- exam durations between one and three hours (longest is 12 hours - neglected)
- exam session of 4 weeks of 6 days from 7 am to 7 pm (assumption)
- 30 rooms with the capacity for 300 students (assumption)

The exam registrations are more or less evenly distributed over all exams.

Compared to the dataset our solution was designed for, the new dataset contains 10-times the number of students, 2.5-times the number of exams, and 6-times the number of exam registrations. This averages to only ~ 3 exams per student per exam session. Students at OST-RJ have, on average, ~ 6.5 exams per exam session – twice as many. When also considering that the exam session at the ETH is 24 days, compared to the 15 days at OST, it shows that the problem is a bit simpler.

When running this dataset, the larger scale becomes clear very fast. The construction phase, which usually takes around 10 minutes, took more than one hour to complete. That being said, an initial solution was found, and solution improvements came in quickly. As the input dataset was created artificially, completing the test run would not have brought any beneficial insights. The main goal was to test if the solver can handle a large-scale dataset and if it is reasonable to apply the constraint solving approach to it. In-depth tests and a real dataset are required conclusively to answer the question regarding the solution's scalability, but the observed results look promising.

5.7 Outlook Constraint Solving

After spending so much time working with a constraint solver, we can paint a pretty good picture of the current state of constraint solvers and how this field will develop. In the current uprising of artificial intelligence in the form of neural networks, one may ask if they will replace constraint solvers in the future. Taking neural networks as the answer to everything is a bit superficial. Constraint solvers or metaheuristics, for that matter, come into play when dealing with an ever-changing, unpredictable environment, like scheduling exams of different durations, attendees, or requirements. Neuronal networks, on the other hand, mimic the human mind. They try to find patterns they then, later on, can apply to new, unknown problems. Therefore a neural network will probably not be able to replace these metaheuristics anytime soon, but a combination of both may be a promising approach.

Metaheuristics choose the next step by picking the best one out of a set of possible steps. The larger the sets are, the slower the algorithm performs. The smaller the sets are, the faster the algorithm performs, with the cost of not always choosing the best possible next step. With the help of neural networks, the selection process could probably be enhanced. Let the neural network find patterns of changes or swaps that historically have shown the most significant improvement. With something like that, the pick-quality could be improved by a lot, without losing too much performance.

This is a hypothesis that we have come up with while working with a constraint solver. It would need to be carefully tested, but could result in an enormous leap forward for the field of constraint solving.

Chapter 6

Conclusion

In our semester project, we showed that the automation of examination timetable creation for OST is possible. In our bachelor's thesis project, we have now built the first big step for making this idea become reality. The now enhanced exam scheduler can create usable exam schedules optimized for the students and includes a wide variety of additional constraints to cover availabilities and particular exam requirements or room features. The model is now ready for a first pilot phase.

As previously described, a comparison with a manually created exam schedule showed that the exam scheduler can create equal or even better schedules than a human could do, while reducing the time to create an exam schedule from several weeks to a few days or even hours.

The powerful UI enables the (human) exam scheduler to create exam schedules with many insights, while still keeping the flexibility to intervene and place particular exams manually. The UI can even be used without the solver to create exam schedules. All of this, without missing out on any of the validations and verifications. In-depth tests by the end-user/customer are now required to make the software ultimately production-ready.

All that progress did not come without any obstacles: Missing test environments needed to be organized and set up, formulas for measuring an exam distribution had to be found, different algorithms had to be evaluated and compared to each other, missing documentation and missing features had to be figured out and worked around. All while, in parallel, designing and implementing a full-fledged frontend for controlling and visualizing everything. Furthermore, running tests always took several hours before getting a usable result and potentially determining that something was wrong and that it needed to be adapted and rerun. Ultimately, we mastered everything and can proudly present the **Automated Exam Scheduler – Version 1.0**.

6.1 Features for Future Releases

Finishing the first version should not be the end of the exam scheduler's journey. Some existing features or constraints may need some adjustment/refinement, or many additional features may be added.

For example, replacing the regular semesters with the sample study plans could yield much better results, or providing further KPIs can give the user an even better understanding of the schedule on hand. The following is our extended but not exhaustive list of ideas of further features to the exam scheduler:

- Replace regular semesters with sample study plans
- KPIs / further insights into the solution directly in the frontend
- Partial locking of exams (only room or only date and time)
- "Ghost-exams" that do not count for the validation
- Possibility to disable certain constraints for specific exams
- Possibility to adjust constraint weights from the UI directly
- Exporting schedules and import them for later use
- Unavailabilities for examiners
- Grouping of exams, for scheduling them at the same time
- Automatic oral exam planning
- Special handling of splitting exams to multiple available computer room
- Create snapshots and clones of schedules
- Expand the area of operation to other universities/campuses

Appendix A

Guest Lecture “Automaten und Sprachen”

Professor Dr. Andreas Müller gives lectures regarding theoretical computer science, including NP-complete problems at the OST. As exam scheduling is also known to be an NP-complete problem, Mr. Müller asked us if we are interested in giving a guest lecture for the students to show them how NP-completeness can be found in the real world and how it can be tackled.

Due to the COVID-19 pandemic, the lecture had to be held online. The presentation itself was recorded, but the students had the chance to ask us questions afterward via the live chat.

Figure A.1: Title Slide

This recordings of those lectures (held in German) can be found on YouTube via this links:

- “AutoSpr Online-Vorlesung 18. Mai 2021, 17:00-17:45”: `https://www.youtube.com/watch?v=0Z5te8HyJB4`
- “AutoSpr: Online-Vorlesung 20. Mai 2021, 14:05-14:50”: `https://www.youtube.com/watch?v=j4mZBqXSBFI`

Appendix B

Code Stats

B.1 Lines of Code (LOC)

Table B.1: Lines of Code (LOC)

File Type	Count	LOC
java	170	7151
java (Test Code)	73	13715
kotlin	20	338
kotlin (Test Code)	12	1244
typescript	160	6009
typescript (Generated)	48	1205
scss	36	885
html	33	1529
Total	(3.7 x of SA) **552**	(3.3 x of SA) **32076**

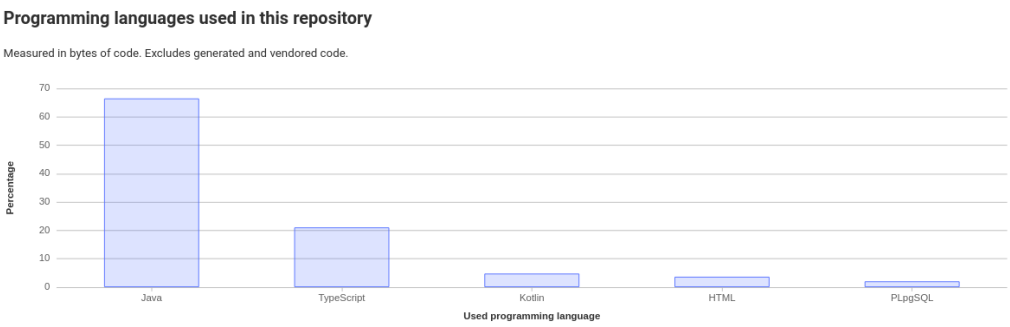


Figure B.1: Used Programming Languages

B.2 Test Coverage

Figure B.2: Passed Tests

81% classes, 77% lines covered in package 'ch.ost.examscheduler'

Element	Class, %	Method, %	Line, %
application	75% (3/4)	68% (13/19)	50% (45/90)
benchmark	0% (0/1)	0% (0/7)	0% (0/31)
config	100% (7/7)	89% (17/19)	72% (66/91)
core	79% (83/105)	72% (513/712)	74% (1361/1825)
domain	58% (7/12)	40% (14/35)	46% (30/64)
interfaces	25% (1/4)	10% (1/10)	4% (1/21)
solvers	94% (50/53)	88% (281/316)	93% (848/911)
utils	100% (2/2)	50% (3/6)	50% (3/6)
ExamSchedulerApplication	100% (1/1)	0% (0/1)	33% (1/3)

Figure B.3: Test Coverage

B.3 Issues and CI/CD

- Issues: 112
- Total: 427 pipelines, Successful: 298 pipelines, Failed: 119 pipelines

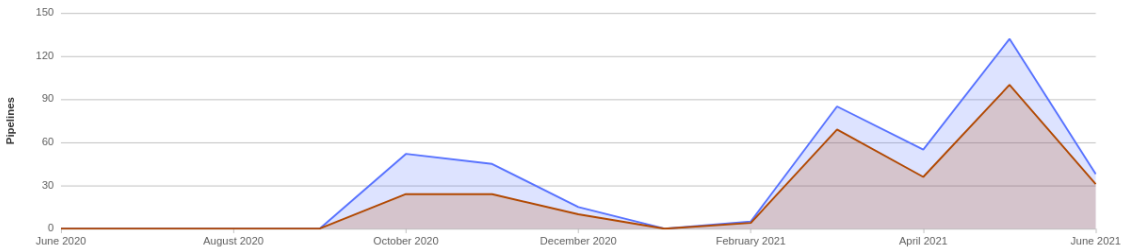


Figure B.4: CI/CD Pipeline Chart

Appendix C

Verification of Cost Function for Optimal Exam Distribution

- Examination Schedules
- Test Results

C.1 Examination Schedules

Description	Examination Schedule																	
	16.1.17	17.1.17	18.1.17	19.1.17	20.1.17	21.1.17	22.1.17	23.1.17	24.1.17	25.1.17	26.1.17	27.1.17	28.1.17	29.1.17	30.1.17	31.1.17	1.2.17	2.2.17	3.2.17
7 Good	12/00 Uhr			10/40 Uhr				9/00 Uhr		15/10 Uhr		8/00 Uhr				11/00 Uhr			10/30 Uhr
7 No free days	12/00 Uhr	10/40 Uhr	9/00 Uhr	15/10 Uhr	10/30 Uhr			10/30 Uhr	9/00 Uhr										
7 No free days worse	14/00 Uhr	8/10 Uhr	14/00 Uhr	9/20 Uhr	14/00 Uhr			16/20 Uhr	10/20 Uhr										
7 Cluster	8/10 Uhr	14/00 Uhr			14/00 Uhr				9/20 Uhr	14/00 Uhr					10/20 Uhr	16/20 Uhr			
7 Cluster worse	14/00 Uhr	8/10 Uhr			14/00 Uhr				14/00 Uhr	9/20 Uhr					16/20 Uhr	10/20 Uhr			
7 Compact but good	12/00 Uhr		10/40 Uhr		9/00 Uhr			15/10 Uhr			8/00 Uhr	11/00 Uhr			10/30 Uhr				
7 Compact but good (slightly better)	12/00 Uhr		10/40 Uhr		9/00 Uhr			15/10 Uhr		8/00 Uhr		11/00 Uhr			10/30 Uhr				
12 Good	8/10 Uhr	14/00 Uhr		8/10 Uhr	14/00 Uhr			8/10 Uhr	14/00 Uhr		8/10 Uhr	14/00 Uhr			8/10 Uhr	14/00 Uhr		8/10 Uhr	14/00 Uhr
12 Bad I		8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr			8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr				8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr	
12 Bad II	8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr					8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr				8/10 Uhr	14/00 Uhr	8/10 Uhr	14/00 Uhr
2 Max spread	8/10 Uhr																		15/00 Uhr
2 No spread				8/10 Uhr	15/00 Uhr														
5 Compact but good	12/00 Uhr		10/40 Uhr					9/00 Uhr		15/10 Uhr		8/00 Uhr							
5 Evenly spread all over	12/00 Uhr				10/40 Uhr					9/00 Uhr					15/10 Uhr				8/00 Uhr
1 Reference								13/00 Uhr											
15 Exams	12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr			12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr			12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr	12/00 Uhr
1 Exam	8/00 Uhr																		
2 Exams	8/00 Uhr			8/00 Uhr															
3 Exams (good)	8/00 Uhr			8/00 Uhr				8/00 Uhr											
3 Exams (bad)	8/00 Uhr		8/00 Uhr		8/00 Uhr														
3 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr																
4 Exams (good)	8/00 Uhr			8/00 Uhr				8/00 Uhr			8/00 Uhr								
4 Exams (bad)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr											
4 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr															
5 Exams (good)	8/00 Uhr			8/00 Uhr				8/00 Uhr			8/00 Uhr				8/00 Uhr				
5 Exams (bad)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr									
5 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr														
6 Exams (good)	8/00 Uhr			8/00 Uhr				8/00 Uhr			8/00 Uhr				8/00 Uhr			8/00 Uhr	
6 Exams (bad)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr							
6 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr											
7 Exams (good)	8/00 Uhr		15/00 Uhr		8/00 Uhr				8/00 Uhr			8/00 Uhr			8/00 Uhr			8/00 Uhr	
7 Exams (bad)	8/00 Uhr		8/00 Uhr	8/00 Uhr				8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr				
7 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr		8/00 Uhr									
8 Exams (good)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr			8/00 Uhr	
8 Exams (bad)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr				
8 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr	8/00 Uhr									
9 Exams (good)	8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr
9 Exams (bad)	8/00 Uhr		8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr		8/00 Uhr	8/00 Uhr	8/00 Uhr				8/00 Uhr			
9 Exams (worst)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr								
10 Exams (good)	8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr			8/00 Uhr		8/00 Uhr		8/00 Uhr
10 Exams (bad)	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr	8/00 Uhr					8/00 Uhr	8/00 Uhr			
10 Exams (worst)	15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr							
11 Exams (good)	8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr			8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr			8/00 Uhr		15/00 Uhr		15/00 Uhr
11 Exams (bad)	15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr	8/00 Uhr					8/00 Uhr	8/00 Uhr	8/00 Uhr		
11 Exams (worst)	15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			8/00 Uhr				
12 Exams (ok)	8/00 Uhr	8/00 Uhr		8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr		8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr		8/00 Uhr	8/00 Uhr
12 Exams (good)	8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr			8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr			8/00 Uhr	15/00 Uhr		8/00 Uhr	15/00 Uhr
12 Exams (bad)	8/00 Uhr	14/00 Uhr	8/00 Uhr	8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr		8/00 Uhr	8/00 Uhr			8/00 Uhr	8/00 Uhr	8/00 Uhr		
12 Exams (worst)	15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			15/00 Uhr	8/00 Uhr	8/00 Uhr	15/00 Uhr	8/00 Uhr			15/00 Uhr	8/00 Uhr			

C.2 Test Results

Description	Initial Approach		Initial Approach		Adhoc	Corrected Optimal D.		Costs from Iterations				Coststs from Iterations (scaled by 1000)			
	Optimal Distance [d]	Costs	Optimal Distance [h]	Costs	Variante	[d]	[h]	1st Iteration	2nd Iteration	3rd Iteration	4th Iteration	1st Iteration	2nd Iteration	3rd Iteration	4th Iteration
7 Good	3.00	0.67	72.00	419.87	0.00	2.00	48.00	0.267	0.025	0.025	0.012	267	25	25	12
7 No free days	3.00	3.33	72.00	1979.10	11.00	2.00	48.00	0.210	0.193	0.193	0.090	210	193	193	90
7 No free days worse	3.00	3.33	72.00	2041.51	11.00	2.00	48.00	0.220	0.200	0.200	0.093	220	200	200	93
7 Cluster	3.00	2.83	72.00	1293.17	3.00	2.00	48.00	0.313	0.111	0.111	0.052	313	111	111	52
7 Cluster worse	3.00	2.83	72.00	2037.14	3.00	2.00	48.00	0.381	0.177	0.177	0.083	381	177	177	83
7 Compact but good	3.00	1.00	72.00	569.20	1.00	2.00	48.00	0.163	0.073	0.073	0.034	163	73	73	34
7 Compact but good (slightly better)	3.00	0.67	72.00	458.53	0.00	2.00	48.00	0.136	0.026	0.026	0.012	136	26	26	12
12 Good	1.64	0.60	39.27	182.39	0.00	1.17	28.00	0.193	0.000	0.000	0.000	193	0	0	0
12 Bad I	1.64	1.01	39.27	471.32	12.00	1.17	28.00	0.244	0.055	0.055	0.044	244	55	55	44
12 Bad II	1.64	1.87	39.27	915.48	12.00	1.17	28.00	0.350	0.055	0.055	0.044	350	55	55	44
2 Max spread	18.00	0.00	432.00	46.69	0.00	7.00	168.00	1.612	0.000	0.000	0.000	1612	0	0	0
2 No spread	18.00	289.00	432.00	160934.69	1.00	7.00	168.00	0.816	0.816	0.486	0.065	816	816	486	65
5 Compact but good	4.50	4.75	108.00	2819.49	0.00	2.80	67.20	0.232	0.133	0.100	0.033	232	133	100	33
5 Evenly spread all over	4.50	0.25	108.00	245.49	0.00	2.80	67.20	0.318	0.000	0.000	0.000	318	0	0	0
1 Reference	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0	0	0	0
15 Exams	1.29	0.49	30.86	282.12	18.00	0.93	22.40	0.224	0.000	0.000	0.000	224	0	0	0
1 Exam	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0	0	0	0
2 Exams	18.00	225.00	432.00	129600.00	0.00	7.00	168.00	0.571	0.571	0.000	0.000	571	571	0	0
3 Exams (good)	9.00	30.50	216.00	17568.00	0.00	4.67	112.00	0.192	0.192	0.000	0.000	192	192	0	0
3 Exams (bad)	9.00	49.00	216.00	28224.00	0.00	4.67	112.00	0.404	0.404	0.141	0.028	404	404	141	28
3 Exams (worst)	9.00	64.00	216.00	36864.00	3.00	4.67	112.00	0.556	0.556	0.424	0.085	556	556	424	85
4 Exams (good)	6.00	7.33	144.00	4224.00	0.00	3.50	84.00	0.082	0.067	0.000	0.000	82	67	0	0
4 Exams (bad)	6.00	13.67	144.00	7872.00	0.00	3.50	84.00	0.208	0.208	0.094	0.025	208	208	94	25
4 Exams (worst)	6.00	25.00	144.00	14400.00	6.00	3.50	84.00	0.412	0.412	0.346	0.092	412	412	346	92
5 Exams (good)	4.50	1.25	108.00	720.00	0.00	2.80	67.20	0.154	0.000	0.000	0.000	154	0	0	0
5 Exams (bad)	4.50	5.25	108.00	3024.00	0.00	2.80	67.20	0.125	0.124	0.087	0.029	125	124	87	29
5 Exams (worst)	4.50	12.25	108.00	7056.00	10.00	2.80	67.20	0.321	0.321	0.300	0.100	321	321	300	100
6 Exams (good)	3.60	0.28	86.40	161.28	0.00	2.33	56.00	0.225	0.000	0.000	0.000	225	0	0	0
6 Exams (bad)	3.60	2.12	86.40	1221.12	0.00	2.33	56.00	0.081	0.057	0.057	0.023	81	57	57	23
6 Exams (worst)	3.60	5.48	86.40	3156.48	10.00	2.33	56.00	0.236	0.229	0.229	0.091	236	229	229	91
7 Exams (good)	3.00	0.50	72.00	304.33	0.00	2.00	48.00	0.223	0.024	0.024	0.011	223	24	24	11
7 Exams (bad)	3.00	1.33	72.00	768.00	1.00	2.00	48.00	0.204	0.083	0.083	0.039	204	83	83	39
7 Exams (worst)	3.00	2.83	72.00	1632.00	10.00	2.00	48.00	0.186	0.167	0.167	0.078	186	167	167	78
8 Exams (good)	2.57	0.27	61.71	152.82	0.00	1.75	42.00	0.181	0.000	0.000	0.000	181	0	0	0
8 Exams (bad)	2.57	0.90	61.71	517.22	3.00	1.75	42.00	0.172	0.087	0.087	0.046	172	87	87	46
8 Exams (worst)	2.57	2.14	61.71	1234.29	13.00	1.75	42.00	0.181	0.150	0.150	0.080	181	150	150	80
9 Exams (good)	2.25	0.19	54.00	108.00	0.00	1.56	37.33	0.186	0.000	0.000	0.000	186	0	0	0
9 Exams (bad)	2.25	1.25	54.00	720.00	6.00	1.56	37.33	0.250	0.089	0.089	0.054	250	89	89	54
9 Exams (worst)	2.25	1.44	54.00	828.00	16.00	1.56	37.33	0.166	0.118	0.118	0.071	166	118	118	71
10 Exams (good)	2.00	0.44	48.00	165.78	0.00	1.40	33.60	0.192	0.012	0.012	0.008	192	12	12	8
10 Exams (bad)	2.00	1.89	48.00	1088.00	12.00	1.40	33.60	0.324	0.084	0.084	0.056	324	84	84	56
10 Exams (worst)	2.00	1.00	48.00	726.11	18.00	1.40	33.60	0.192	0.119	0.119	0.079	192	119	119	79
11 Exams (good)	1.80	0.56	43.20	171.78	0.00	1.27	30.55	0.194	0.000	0.000	0.000	194	0	0	0
11 Exams (bad)	1.80	1.68	43.20	1009.26	12.00	1.27	30.55	0.332	0.079	0.079	0.058	332	79	79	58
11 Exams (worst)	1.80	0.80	43.20	589.18	16.00	1.27	30.55	0.229	0.094	0.094	0.069	229	94	94	69
12 Exams (ok)	1.64	0.60	39.27	342.74	0.00	1.17	28.00	0.233	0.032	0.032	0.025	233	32	32	25
12 Exams (good)	1.64	0.60	39.27	158.49	0.00	1.17	28.00	0.186	0.000	0.000	0.000	186	0	0	0
12 Exams (bad)	1.64	0.64	39.27	377.85	9.00	1.17	28.00	0.217	0.045	0.045	0.036	217	45	45	36
12 Exams (worst)	1.64	0.67	39.27	567.30	15.00	1.17	28.00	0.248	0.082	0.082	0.066	248	82	82	66

Appendix D

User Stories

D.1 Completed Stories

D.1.1 Unavailable time periods of rooms

As an examination planner, I want to define the time periods a certain room is unavailable so that no exams are scheduled in this room during these periods.

D.1.2 Consider AC in spring semesters

As an examination planner, I want to define whether I plan the exams for a spring semester or a fall semester so that the exam scheduler considers the air conditioning for spring semesters.

D.1.3 Schedule some exams manually in advance

As an examination planner, I want to schedule some exams manually before the scheduler starts planning all other exams so that exams that are too complicated to schedule automatically or are special cases can be set by hand.

D.1.4 Support mulitple exam schedules for a semester

As an examination planner, I want to create multiple exam schedules for the same semester, so that I can work on more than one schedule and compare them.

D.1.5 Support history of exam schedules

As an examination planner, I want to view previous exam schedules, e.g. from last year, so that I can use that information the schedule the current exams.

D.1.6 Support scheduling exams in multiple rooms

As an examination planner I want the application to be able to schedule exams in to multiple rooms if a single room has no space for all student so that I can schedule all exams automatically.

Details

- An exam with up to 100 students can be scheduled into 1 room.
- An exam with more than 100 students can be scheduled into up to 2 rooms.
- An exam with more than 200 students can be scheduled into up to 3 rooms.

D.1.7 Distribute exams of regular semesters evenly

As an examination planner, I want the application to distribute the exams of a regular semester as evenly as possible over the examination session so that I can optimize the exam schedule for a regular semester.

D.1.8 Distribute exams of individual students evenly

As an examination planner, I want the application to distribute the exams of an individual students as evenly as possible over the examination session so that I can optimize the exam schedule for an individual semster.

D.1.9 Consider computer rooms

As an examination planner I want the application to be able to schedule exams that need a computer in to a computer room so that I can schedule all exams automatically.

D.1.10 Schedule some exams on same time and in same room

As an examination planer, I want to define group of exams that must take place at the same time and in the same room so that more exams can be scheduled and students can not change information about the exam.

D.1.11 Avoid scheduling exams during lunch

As an examination planner, I want to avoid that exams are scheduled over noon so that students can eat lunch at a proper lunch time

D.1.12 Start/resume, stop/pause scheduler

As an examination planner, I want to stop the solving porcess so that I can continue it at a later time.

D.1.13 Implement multiple timetables with corresponding data

As a user, I want to be able to create multiple exam schedules with different input data sets.

D.1.14 Add overview page for listing all available exam schedules

As an examination planner, I want to have an overview of all exam schedules currently present in the application to be able to access and manage them.

D.1.15 Implement basic score details view

As an examination planner, I want to see why the score is the score that it is, in order to understand and adjust the solving process.

D.1.16 Add simple way to set a fixed room and dateTime to exam

As an examination planner, I want to be able to fix certain exams to a specific time and room in advance.

D.1.17 Add possibility to start/stop solving in the frontend

Follow-up of subsection D.1.12

As an examination scheduler, I want to start and stop the solving process directly from the front end.

D.1.18 Create Exam Schedule creation wizard

As an examination scheduler, I want to be able to create a new exam scheduler directly from the front end.

D.1.19 Add option for exporting an exam schedule

As an examination planner, I want to export a solved schedule to JSON or CSV directly from the frontend.

D.1.20 Add option to search for an exam

As an examination planner, I want to search for specific exams, in order for me to verify the end score.

D.1.21 Create exam details view

As an examination planner, I want to see the details of an exam (Name, description, time, room, students, ...)

D.1.22 Add option to delete timetables

As an examination planner, I want to delete the timetables I created directly from the frontend.

D.1.23 Lock/Unlock a timetable

As an exam scheduler I want to lock/unlock a timetable (make read-only).

D.1.24 Schedule exams in air conditioned room in summer

As an exam scheduler, I want the option to only schedule exams at the afternoon in air conditioned rooms in summer.

D.1.25 Extract and show score details

As an exam scheduler, I want to get insight on the score details to better understand the generated results.

D.1.26 Get notified / update solution if a new one is found

As an exam scheduler, I want to get notified in some way that a new solution is available.

D.1.27 Lock/Unlock a timetable

As an examination planer, I want to define group of exams that must take place at the same time and in the same room so that more exams can be scheduled and students can not change information about the exam.

D.1.28 Restrict access to frontend and api for unauthorized users

As a university, I don't want that any data, especially student specific data, can be accesses by everyone. As an exam scheduler, I don't want that everyone is able to adjust the data without being authenticated.

D.2 Open / Partially implemented / Future Stories

D.2.1 Unavailable time periods for examiners

As an examination planner, I want to define the time periods a certain examiner is unavailable so that no of his/her exams are scheduled during these periods.

D.2.2 Distribute exams of a examiner evenly

As an examination planner, I want the application to distribute the exams of an examiner as evenly as possible over the examination session so that I can optimize the exam schedule for an examiner.

Will not be implemented, as it would result in a high negative impact. A fine granular definition would be necessary if this is a feature that is needed.

D.2.3 Add option to visualize exam schedule based on JSON file

Based on subsection D.1.19

As an examination planner, I want to visualize a previously exported schedule directly in the frontend.

D.2.4 Add autocompletion

As an examination planner, I want to have auto-completion in the filter fields.

D.2.5 Add solving machine stats view

As an examination planner, I want to see the current stats of the solving machine, in order for me to verify the correct operation.

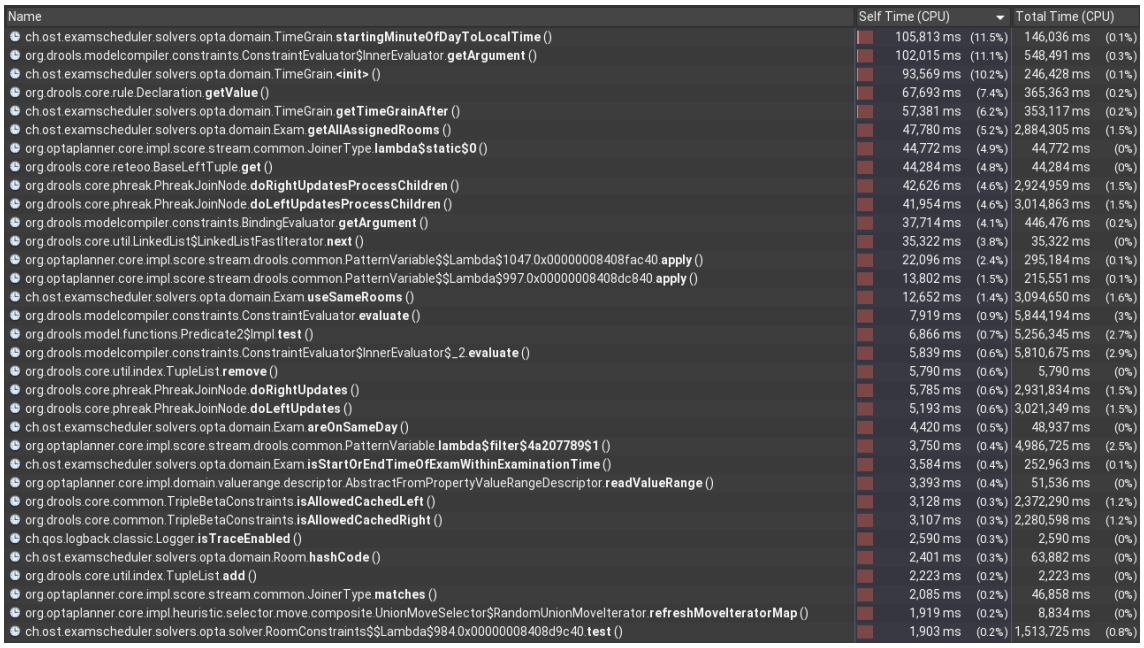
Appendix E

Runtime Profiling

This appendix shows the expanded view of the VisualVM profiling of the Exam Scheduler application. It is seperated into Self Time and Total Time.

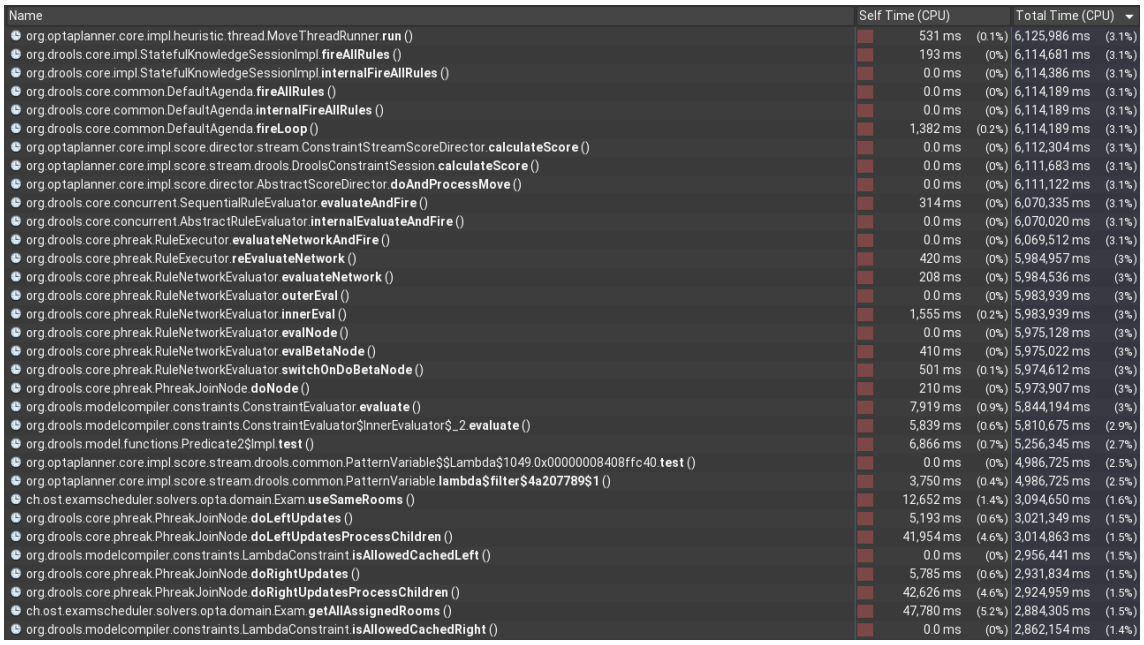
For visibility reasons and ease of comparison, the screenshots of the individual runs are displayed on separate pages.

E.1 Profiling Run 1 - Extended and Unfiltered



Name	Self Time (CPU)		Total Time (CPU)	
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**startingMinuteOfDayToLocalTime** ()	105,813 ms	(11.5%)	146,036 ms	(0.1%)
org.drools.modelcompiler.constraints.ConstraintEvaluator$InnerEvaluator.**getArgument** ()	102,015 ms	(11.1%)	548,491 ms	(0.3%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**<init>** ()	93,569 ms	(10.2%)	246,428 ms	(0.1%)
org.drools.core.rule.Declaration.**getValue** ()	67,693 ms	(7.4%)	365,363 ms	(0.2%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**getTimeGrainAfter** ()	57,381 ms	(6.2%)	353,117 ms	(0.2%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	47,780 ms	(5.2%)	2,884,305 ms	(1.5%)
org.optaplanner.core.impl.score.stream.common.JoinerType.**lambda$static$0** ()	44,772 ms	(4.9%)	44,772 ms	(0%)
org.drools.core.reteoo.BaseLeftTuple.**get** ()	44,284 ms	(4.8%)	44,284 ms	(0%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdatesProcessChildren** ()	42,626 ms	(4.6%)	2,924,959 ms	(1.5%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdatesProcessChildren** ()	41,954 ms	(4.6%)	3,014,863 ms	(1.5%)
org.drools.modelcompiler.constraints.BindingEvaluator.**getArgument** ()	37,714 ms	(4.1%)	446,476 ms	(0.2%)
org.drools.core.util.LinkedList$LinkedListFastIterator.**next** ()	35,322 ms	(3.8%)	35,322 ms	(0%)
org.optaplanner.core.impl.score.stream.drools.common.PatternVariable$$Lambda$1047.0x00000008408fac40.**apply** ()	22,096 ms	(2.4%)	295,184 ms	(0.1%)
org.optaplanner.core.impl.score.stream.drools.common.PatternVariable$$Lambda$997.0x00000008408dc840.**apply** ()	13,802 ms	(1.5%)	215,551 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**useSameRooms** ()	12,652 ms	(1.4%)	3,094,650 ms	(1.6%)
org.drools.modelcompiler.constraints.ConstraintEvaluator.**evaluate** ()	7,919 ms	(0.9%)	5,844,194 ms	(3%)
org.drools.model.functions.Predicate2$Impl.**test** ()	6,866 ms	(0.7%)	5,256,345 ms	(2.7%)
org.drools.modelcompiler.constraints.ConstraintEvaluator$InnerEvaluator$_2.**evaluate** ()	5,839 ms	(0.6%)	5,810,675 ms	(2.9%)
org.drools.core.util.index.TupleList.**remove** ()	5,790 ms	(0.6%)	5,790 ms	(0%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdates** ()	5,785 ms	(0.6%)	2,931,834 ms	(1.5%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdates** ()	5,193 ms	(0.6%)	3,021,349 ms	(1.5%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**areOnSameDay** ()	4,420 ms	(0.5%)	48,937 ms	(0%)
org.optaplanner.core.impl.score.stream.drools.common.PatternVariable.**lambda$filter$4a207789$1** ()	3,750 ms	(0.4%)	4,986,725 ms	(2.5%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**isStartOrEndTimeOfExamWithinExaminationTime** ()	3,584 ms	(0.4%)	252,963 ms	(0.1%)
org.optaplanner.core.impl.domain.valuerange.descriptor.AbstractFromPropertyValueRangeDescriptor.**readValueRange** ()	3,393 ms	(0.4%)	51,536 ms	(0%)
org.drools.core.common.TripleBetaConstraints.**isAllowedCachedLeft** ()	3,128 ms	(0.3%)	2,372,290 ms	(1.2%)
org.drools.core.common.TripleBetaConstraints.**isAllowedCachedRight** ()	3,107 ms	(0.3%)	2,280,598 ms	(1.2%)
ch.qos.logback.classic.Logger.**isTraceEnabled** ()	2,590 ms	(0.3%)	2,590 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Room.**hashCode** ()	2,401 ms	(0.3%)	63,882 ms	(0%)
org.drools.core.util.index.TupleList.**add** ()	2,223 ms	(0.2%)	2,223 ms	(0%)
org.optaplanner.core.impl.score.stream.common.JoinerType.**matches** ()	2,085 ms	(0.2%)	46,858 ms	(0%)
org.optaplanner.core.impl.heuristic.selector.move.composite.UnionMoveSelector$RandomUnionMoveIterator.**refreshMoveIteratorMap** ()	1,919 ms	(0.2%)	8,834 ms	(0%)
ch.ost.examscheduler.solvers.opta.solver.RoomConstraints$$Lambda$984.0x00000008408d9c40.**test** ()	1,903 ms	(0.2%)	1,513,725 ms	(0.8%)

Figure E.1: Sorted by Self Time (Total Execution Time: 1,050 seconds = 17.5 minutes)



Name	Self Time (CPU)		Total Time (CPU)	
org.optaplanner.core.impl.heuristic.thread.MoveThreadRunner.**run** ()	531 ms	(0.1%)	6,125,986 ms	(3.1%)
org.drools.core.impl.StatefulKnowledgeSessionImpl.**fireAllRules** ()	193 ms	(0%)	6,114,681 ms	(3.1%)
org.drools.core.impl.StatefulKnowledgeSessionImpl.**internalFireAllRules** ()	0.0 ms	(0%)	6,114,386 ms	(3.1%)
org.drools.core.common.DefaultAgenda.**fireAllRules** ()	0.0 ms	(0%)	6,114,189 ms	(3.1%)
org.drools.core.common.DefaultAgenda.**internalFireAllRules** ()	0.0 ms	(0%)	6,114,189 ms	(3.1%)
org.drools.core.common.DefaultAgenda.**fireLoop** ()	1,382 ms	(0.2%)	6,114,189 ms	(3.1%)
org.optaplanner.core.impl.score.director.stream.ConstraintStreamScoreDirector.**calculateScore** ()	0.0 ms	(0%)	6,112,304 ms	(3.1%)
org.optaplanner.core.impl.score.stream.drools.DroolsConstraintSession.**calculateScore** ()	0.0 ms	(0%)	6,111,683 ms	(3.1%)
org.optaplanner.core.impl.score.director.AbstractScoreDirector.**doAndProcessMove** ()	0.0 ms	(0%)	6,111,122 ms	(3.1%)
org.drools.core.concurrent.SequentialRuleEvaluator.**evaluateAndFire** ()	314 ms	(0%)	6,070,335 ms	(3.1%)
org.drools.core.concurrent.AbstractRuleEvaluator.**internalEvaluateAndFire** ()	0.0 ms	(0%)	6,070,020 ms	(3.1%)
org.drools.core.phreak.RuleExecutor.**evaluateNetworkAndFire** ()	0.0 ms	(0%)	6,069,512 ms	(3.1%)
org.drools.core.phreak.RuleExecutor.**reEvaluateNetwork** ()	420 ms	(0%)	5,984,957 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**evaluateNetwork** ()	208 ms	(0%)	5,984,536 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**outerEval** ()	0.0 ms	(0%)	5,983,939 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**innerEval** ()	1,555 ms	(0.2%)	5,983,939 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**evalNode** ()	0.0 ms	(0%)	5,975,128 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**evalBetaNode** ()	410 ms	(0%)	5,975,022 ms	(3%)
org.drools.core.phreak.RuleNetworkEvaluator.**switchOnDoBetaNode** ()	501 ms	(0.1%)	5,974,612 ms	(3%)
org.drools.core.phreak.PhreakJoinNode.**doNode** ()	210 ms	(0%)	5,973,907 ms	(3%)
org.drools.modelcompiler.constraints.ConstraintEvaluator.**evaluate** ()	7,919 ms	(0.9%)	5,844,194 ms	(3%)
org.drools.modelcompiler.constraints.ConstraintEvaluator$InnerEvaluator$_2.**evaluate** ()	5,839 ms	(0.6%)	5,810,675 ms	(2.9%)
org.drools.model.functions.Predicate2$Impl.**test** ()	6,866 ms	(0.7%)	5,256,345 ms	(2.7%)
org.optaplanner.core.impl.score.stream.drools.common.PatternVariable$$Lambda$1049.0x00000008408ffc40.**test** ()	0.0 ms	(0%)	4,986,725 ms	(2.5%)
org.optaplanner.core.impl.score.stream.drools.common.PatternVariable.**lambda$filter$4a207789$1** ()	3,750 ms	(0.4%)	4,986,725 ms	(2.5%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**useSameRooms** ()	12,652 ms	(1.4%)	3,094,650 ms	(1.6%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdates** ()	5,193 ms	(0.6%)	3,021,349 ms	(1.5%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdatesProcessChildren** ()	41,954 ms	(4.6%)	3,014,863 ms	(1.5%)
org.drools.modelcompiler.constraints.LambdaConstraint.**isAllowedCachedLeft** ()	0.0 ms	(0%)	2,956,441 ms	(1.5%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdates** ()	5,785 ms	(0.6%)	2,931,834 ms	(1.5%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdatesProcessChildren** ()	42,626 ms	(4.6%)	2,924,959 ms	(1.5%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	47,780 ms	(5.2%)	2,884,305 ms	(1.5%)
org.drools.modelcompiler.constraints.LambdaConstraint.**isAllowedCachedRight** ()	0.0 ms	(0%)	2,862,154 ms	(1.4%)

Figure E.2: Sorted by Total Time (Total Execution Time: 1,050 seconds = 17.5 minutes)

E.2 Profiling Run Final - Extended and Unfiltered

Name	Self Time (CPU)		Total Time (CPU)	
org.drools.modelcompiler.constraints.BindingEvaluator.**getArgument** ()	121,344 ms	(11.6%)	149,070 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.model.TimeGrainsCache.**dateToCacheIndex** ()	100,922 ms	(9.7%)	100,922 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getStartingDateTime** ()	72,244 ms	(6.9%)	72,863 ms	(0.1%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdatesProcessChildren** ()	60,013 ms	(5.7%)	936,649 ms	(0.9%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdatesProcessChildren** ()	51,123 ms	(4.9%)	998,687 ms	(1%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrainCacheProxy.**getTimeGrainAfter** ()	48,562 ms	(4.7%)	149,485 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**lambda$updateTotalCostsAndExamCostProportionMap$1** ()	44,134 ms	(4.2%)	44,134 ms	(0%)
org.optaplanner.core.impl.score.stream.common.JoinerType.**lambda$static$0** ()	42,153 ms	(4%)	42,153 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getAllAssignedRooms** ()	38,710 ms	(3.7%)	46,154 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**calculateExamCostProportionMap** ()	36,854 ms	(3.5%)	131,288 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**getTimeGrainDistancesOfSuccessiveExams** ()	30,064 ms	(2.9%)	39,202 ms	(0%)
org.drools.core.rule.Declaration.**getValue** ()	29,102 ms	(2.8%)	29,390 ms	(0%)
org.drools.core.util.LinkedList$LinkedListFastIterator.**next** ()	25,368 ms	(2.4%)	25,368 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**getTimeGrainDistanceOfExams** ()	24,795 ms	(2.4%)	24,795 ms	(0%)
org.optaplanner.core.impl.score.stream.drools.common.DirectPatternVariable.**lambda$filterForJoin$40b1cdec$1** ()	20,600 ms	(2%)	484,264 ms	(0.5%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**recalculateCosts** ()	17,604 ms	(1.7%)	2,255,060 ms	(2.3%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrainCacheProxy.**getEpochGrainIndex** ()	15,508 ms	(1.5%)	15,508 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**calculateTotalCosts** ()	14,097 ms	(1.4%)	521,476 ms	(0.5%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**isAfter** ()	12,883 ms	(1.2%)	12,883 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**getEndingTimeGrain** ()	10,702 ms	(1%)	160,337 ms	(0.2%)
ch.ost.examscheduler.solvers.opta.domain.TimeGrain.**equals** ()	8,041 ms	(0.8%)	8,041 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**areOnSameDay** ()	7,184 ms	(0.7%)	71,275 ms	(0.1%)
ch.ost.examscheduler.solvers.opta.domain.Student.**lambda$getExamsOnDay$3** ()	5,885 ms	(0.6%)	24,048 ms	(0%)
org.drools.modelcompiler.constraints.ConstraintEvaluator.**evaluate** ()	5,429 ms	(0.5%)	1,862,204 ms	(1.9%)
org.drools.core.phreak.PhreakJoinNode.**doRightUpdates** ()	5,319 ms	(0.5%)	944,696 ms	(0.9%)
org.drools.modelcompiler.consequence.FactHandleLookup.**create** ()	4,562 ms	(0.4%)	4,562 ms	(0%)
org.drools.core.phreak.PhreakJoinNode.**doLeftUpdates** ()	4,445 ms	(0.4%)	1,004,370 ms	(1%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**isStartOrEndTimeOfExamWithinExaminationTime** ()	4,321 ms	(0.4%)	159,552 ms	(0.2%)
org.drools.core.common.DefaultAgenda.**fireLoop** ()	4,182 ms	(0.4%)	2,332,468 ms	(2.3%)
org.drools.core.util.index.TupleList.**remove** ()	4,124 ms	(0.4%)	4,124 ms	(0%)
org.drools.modelcompiler.consequence.LambdaConsequence.**getOriginalFactHandle** ()	3,962 ms	(0.4%)	3,962 ms	(0%)
org.drools.core.util.index.TupleList.**add** ()	3,936 ms	(0.4%)	3,936 ms	(0%)
ch.qos.logback.classic.Logger.**isTraceEnabled** ()	3,752 ms	(0.4%)	3,752 ms	(0%)
ch.ost.examscheduler.solvers.opta.domain.Student.**getCostsForOptimalExamDistribution** ()	3,636 ms	(0.3%)	21,233 ms	(0%)

Figure E.3: Sorted by Self Time (Total Execution Time: 1,050 seconds = 17.5 minutes)

Name	Self Time (CPU)		Total Time (CPU)	
org.optaplanner.core.impl.heuristic.thread.MoveThreadRunner.**run** ()	352 ms	(0%)	4,603,662 ms	(4.6%)
org.optaplanner.core.impl.score.director.AbstractScoreDirector.**doAndProcessMove** ()	0.0 ms	(0%)	4,312,853 ms	(4.3%)
org.optaplanner.core.impl.score.director.stream.DroolsConstraintStreamScoreDirector.**calculateScore** ()	0.0 ms	(0%)	2,333,731 ms	(2.3%)
org.drools.core.impl.StatefulKnowledgeSessionImpl.**fireAllRules** ()	59.3 ms	(0%)	2,333,386 ms	(2.3%)
org.drools.core.impl.StatefulKnowledgeSessionImpl.**internalFireAllRules** ()	0.0 ms	(0%)	2,333,232 ms	(2.3%)
org.drools.core.common.DefaultAgenda.**fireAllRules** ()	0.0 ms	(0%)	2,333,070 ms	(2.3%)
org.drools.core.common.DefaultAgenda.**internalFireAllRules** ()	0.0 ms	(0%)	2,332,468 ms	(2.3%)
org.drools.core.common.DefaultAgenda.**fireLoop** ()	4,182 ms	(0.4%)	2,332,468 ms	(2.3%)
org.optaplanner.core.impl.heuristic.move.AbstractMove.**doMove** ()	186 ms	(0%)	2,272,145 ms	(2.3%)
org.optaplanner.core.impl.heuristic.selector.move.generic.ChangeMove.**doMoveOnGenuineVariables** ()	0.0 ms	(0%)	2,271,222 ms	(2.3%)
org.optaplanner.core.impl.domain.variable.descriptor.VariableDescriptor.**setValue** ()	113 ms	(0%)	2,260,541 ms	(2.3%)
org.optaplanner.core.impl.domain.common.accessor.ReflectionBeanPropertyMemberAccessor.**executeSetter** ()	0.0 ms	(0%)	2,260,254 ms	(2.3%)
ch.ost.examscheduler.solvers.opta.domain.Exam.**setStartingTimeGrain** ()	60.9 ms	(0%)	2,260,079 ms	(2.3%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**recalculateCosts** ()	17,604 ms	(1.7%)	2,255,060 ms	(2.3%)
org.drools.core.concurrent.SequentialRuleEvaluator.**evaluateAndFire** ()	0.0 ms	(0%)	2,240,781 ms	(2.2%)
org.drools.core.concurrent.AbstractRuleEvaluator.**internalEvaluateAndFire** ()	751 ms	(0.1%)	2,239,096 ms	(2.2%)
ch.ost.examscheduler.solvers.opta.domain.cost_functions.OptimalExamDistributionCostFunction.**updateTotalCostsAndExamCostProportionMap** ()	2,224 ms	(0.2%)	2,237,456 ms	(2.2%)
org.drools.core.phreak.RuleExecutor.**evaluateNetworkAndFire** ()	2,858 ms	(0.3%)	2,235,363 ms	(2.2%)
ch.ost.examscheduler.solvers.opta.domain.Exam$$Lambda$2042.0x0000000801888450.**accept** ()	0.0 ms	(0%)	2,176,478 ms	(2.2%)
ch.ost.examscheduler.solvers.opta.domain.Student.**recalculateCostsForOptimalExamDistribution** ()	423 ms	(0%)	2,176,478 ms	(2.2%)
org.drools.core.phreak.RuleExecutor.**reEvaluateNetwork** ()	633 ms	(0.1%)	1,998,074 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**evaluateNetwork** ()	288 ms	(0%)	1,997,441 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**outerEval** ()	216 ms	(0%)	1,995,146 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**innerEval** ()	3,530 ms	(0.3%)	1,994,930 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**evalNode** ()	179 ms	(0%)	1,986,378 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**evalBetaNode** ()	583 ms	(0.1%)	1,986,133 ms	(2%)
org.drools.core.phreak.RuleNetworkEvaluator.**switchOnDoBetaNode** ()	491 ms	(0%)	1,985,550 ms	(2%)
org.drools.core.phreak.PhreakJoinNode.**doNode** ()	0.0 ms	(0%)	1,984,535 ms	(2%)
org.drools.modelcompiler.constraints.ConstraintEvaluator.**evaluate** ()	5,429 ms	(0.5%)	1,862,204 ms	(1.9%)
org.drools.modelcompiler.constraints.ConstraintEvaluator$InnerEvaluator$_2.**evaluate** ()	3,443 ms	(0.3%)	1,807,702 ms	(1.8%)
org.drools.model.functions.Predicate2$Impl.**test** ()	3,482 ms	(0.3%)	1,652,873 ms	(1.7%)
org.optaplanner.core.impl.heuristic.move.CompositeMove.**doMove** ()	128 ms	(0%)	1,485,642 ms	(1.5%)
org.optaplanner.core.impl.score.stream.drools.common.DirectPatternVariable$$Lambda$1105.0x000000080142fce8.**test** ()	0.0 ms	(0%)	1,165,126 ms	(1.2%)
org.optaplanner.core.impl.score.stream.drools.common.DirectPatternVariable.**lambda$filter$59212492$1** ()	1,993 ms	(0.2%)	1,165,126 ms	(1.2%)

Figure E.4: Sorted by Total Time (Total Execution Time: 1,050 seconds = 17.5 minutes)

Appendix F

Benchmarking

F.1 Benchmark Configuration - Test Plan

Nr	RESULT SCORE	DURATION FOR RESULT	unimprovedTimeSpentLimit	constraintStreamImplType	constructionHeuristicType	localSearch	acceptedCountLimit	entityTabuRatio	
1			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~HILL_CLIMBING~~	~~1000~~	-	
2			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~TABU_SEARCH~~	~~1000~~	~~0.02~~	
3			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~SIMULATED_ANNEALING~~	~~4~~	-	~~SimulatedAnnealingStartingTemperature=200hard/400soft~~
4			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~LATE_ACCEPTANCE~~	~~1~~	-	~~LateAcceptanceSize=400~~
5			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~GREAT_DELUGE~~	~~1~~	-	~~GreatDelugeWaterLevelIncrementRatio=0.00000005~~
6			~~6 hours~~	~~BAVET~~	~~FIRST_FIT~~	~~HILL_CLIMBING (step counting)~~	~~1~~	-	~~StepCountingHillClimbingSize=400~~
TAG: benchmark-configuration-1									
1	-11hard/-7785soft	43 min	15 min	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	1	-	LateAcceptanceSize=400
2	0hard/-2123soft	53 min	15 min	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	4	-	LateAcceptanceSize=400
3	ERROR	-	15 min	BAVET	ENTITY_FROM_QUEUE (Default)	SIMULATED_ANNEALING	4	-	SimulatedAnnealingStartingTemperature=200hard/400soft
4	-33hard/-9908soft	56 min	15 min	BAVET	ENTITY_FROM_QUEUE (Default)	GREAT_DELUGE	1	-	GreatDelugeWaterLevelIncrementRatio=0.00000005
5	ERROR	-	15 min	BAVET	ENTITY_FROM_QUEUE (Default)	DEFAULT (LAHC)	Default: 1	-	LateAcceptanceSize=400
TAG: benchmark-configuration-2									
1	-17hard/-8079soft	1h 3m	30 min	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	1	-	LateAcceptanceSize=400
2	0hard/-2446soft	1h 13m	30 min	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	4	-	LateAcceptanceSize=400
3	-3hard/-6304soft	1h 16m	30 min	BAVET	ENTITY_FROM_QUEUE (Default)	SCHC	1	-	StepCountingHillClimbingSize=400
4	ERROR		30 min	BAVET	ENTITY_FROM_QUEUE (Default)	SIMULATED_ANNEALING	4	-	SimulatedAnnealingStartingTemperature=200hard/400soft
5	ERROR		30 min	BAVET	ENTITY_FROM_QUEUE (Default)	GREAT_DELUGE	1	-	GreatDelugeWaterLevelIncrementRatio=0.00000005
6	0hard/-4168soft	1h 58m	30 min	BAVET	ENTITY_FROM_QUEUE (Default)	DEFAULT (LAHC)	Default: 1	-	LateAcceptanceSize=400
TAG: benchmark-configuration-3									
1	-1hard/-76082soft	2d 11h 59m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	SIMULATED_ANNEALING	4	-	SimulatedAnnealingStartingTemperature=200hard/400soft
2	-45hard/-87649soft	4h 35m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	GREAT_DELUGE	1	-	GreatDelugeWaterLevelIncrementRatio=0.00000005
3	-80hard/-80963soft	4h 57m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	1	-	LateAcceptanceSize=400
4	0hard/-66086soft	5h 30m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	LAHC	4	-	LateAcceptanceSize=400
5	0hard/-63643soft	14h 2m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	SCHC	1	-	StepCountingHillClimbingSize=400
6	-1hard/-61867soft	5h 34m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	SCHC	4	-	StepCountingHillClimbingSize=400
7	0hard/-76924soft	6h 55m	4 hours	BAVET	ENTITY_FROM_QUEUE (Default)	DEFAULT (LAHC)	Default: 1	-	LateAcceptanceSize=400
TAG: benchmark-configuration-4									

Figure F.1: Benchmarking Test Plan

F.2 Benchmark Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark xmlns="https://www.optaplanner.org/xsd/benchmark"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="https://www.optaplanner.org/xsd/benchmark https://www.optaplanner.org/xsd/benchmark/benchmark.xsd">

    <benchmarkDirectory>benchmark-data/results/exam-schedule</benchmarkDirectory>
    <parallelBenchmarkCount>1</parallelBenchmarkCount>
    <warmUpSecondsSpentLimit>30</warmUpSecondsSpentLimit>

    <inheritedSolverBenchmark>
        <solver>
            <moveThreadCount>AUTO</moveThreadCount>
            <solutionClass>ch.ost.examscheduler.solvers.opta.domain.ExamTimetable</solutionClass>
            <entityClass>ch.ost.examscheduler.solvers.opta.domain.Exam</entityClass>
            <scoreDirectorFactory>
                <constraintProviderClass>
                    ch.ost.examscheduler.solvers.opta.solver.constraints.TimeTableConstraintProvider
                </constraintProviderClass>
                <constraintStreamImplType>DROOLS</constraintStreamImplType>
                <initializingScoreTrend>ONLY_DOWN/ONLY_DOWN</initializingScoreTrend>
            </scoreDirectorFactory>
```

```
            <termination>
                <unimprovedHoursSpentLimit>4</unimprovedHoursSpentLimit>
            </termination>
        </solver>
        <subSingleCount>1</subSingleCount> <!--number of times each single benchmark run is executed-->
    </inheritedSolverBenchmark>

    <solverBenchmark>
        <name>Simulated Annealing</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <simulatedAnnealingStartingTemperature>200hard/400soft</simulatedAnnealingStartingTemperature ↵
>
                </acceptor>
                <forager>
                    <acceptedCountLimit>4</acceptedCountLimit>
                </forager>
            </localSearch>
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>GREAT_DELUGE</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <greatDelugeWaterLevelIncrementRatio>0.00000005</greatDelugeWaterLevelIncrementRatio>
                </acceptor>
                <forager>
                    <acceptedCountLimit>1</acceptedCountLimit>
                </forager>
            </localSearch>
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>LAHC (Values from Default)</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <lateAcceptanceSize>400</lateAcceptanceSize>
                </acceptor>
                <forager>
                    <acceptedCountLimit>1</acceptedCountLimit>
                </forager>
            </localSearch>
```

```
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>LAHC - Modified (Accepted Count Limit - 4)</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <lateAcceptanceSize>400</lateAcceptanceSize>
                </acceptor>
                <forager>
                    <acceptedCountLimit>4</acceptedCountLimit>
                </forager>
            </localSearch>
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>SCHC - (Accepted Count Limit - 1)</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <stepCountingHillClimbingSize>400</stepCountingHillClimbingSize>
                </acceptor>
                <forager>
                    <acceptedCountLimit>1</acceptedCountLimit>
                </forager>
            </localSearch>
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>SCHC - (Accepted Count Limit - 4)</name>
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
                <acceptor>
                    <stepCountingHillClimbingSize>400</stepCountingHillClimbingSize>
                </acceptor>
                <forager>
                    <acceptedCountLimit>4</acceptedCountLimit>
                </forager>
            </localSearch>
        </solver>
    </solverBenchmark>

    <solverBenchmark>
        <name>DEFAULT (LAHC)</name>
```

```
        <solver>
            <constructionHeuristic/>
            <localSearch>
                <unionMoveSelector>
                    <cacheType>PHASE</cacheType>
                    <changeMoveSelector>
                        <filterClass>
                            ch.ost.examscheduler.solvers.opta.solver.filters.AllRoomsUnassignedChangeMoveFilter
                        </filterClass>
                    </changeMoveSelector>
                </unionMoveSelector>
            </localSearch>
        </solver>
    </solverBenchmark>
</plannerBenchmark>
```

Listing F.1: OptaPlanner Benchmark Configuration

F.3 Benchmark Details

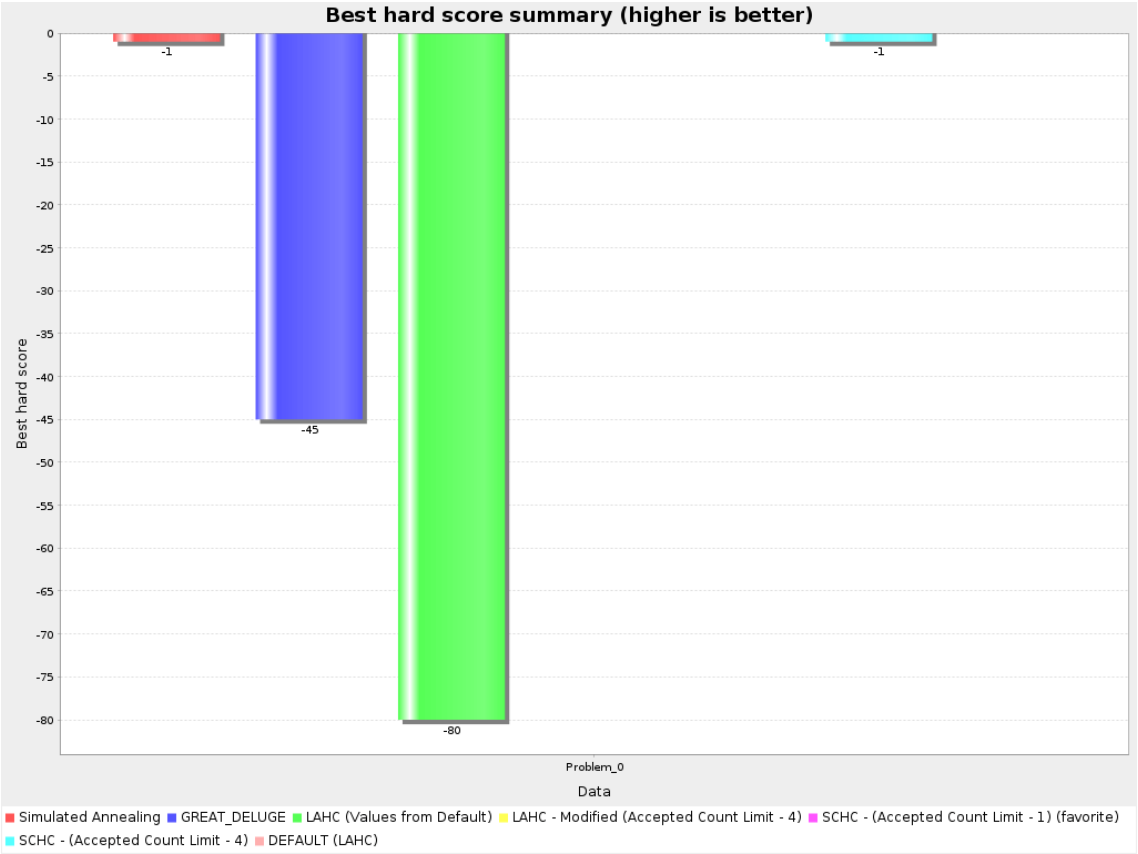


Figure F.2: Benchmark Results – Hard Constraints

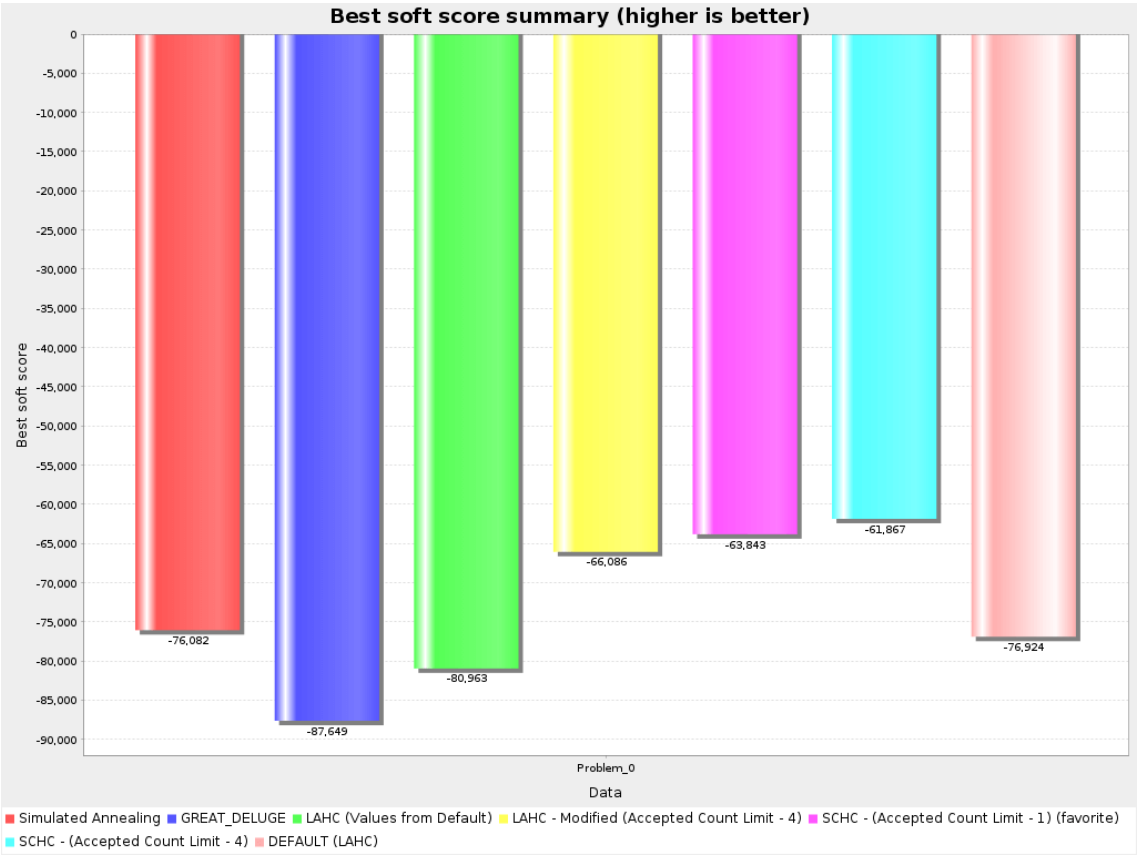


Figure F.3: Benchmark Results – Soft Constraints

Glossary

Angular Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. https://angular.io/ 44

C4 model C4 model is a lean graphical notation technique for modelling the architecture of software systems. It is based on a structural decomposition of a system into containers and components and relies on existing modelling techniques such as the Unified Modelling Language (UML) or Entity Relation Diagrams (ERD) for the more detailed decomposition of the architectural building blocks. [Wikipedia] 84

Constraint Optimization Problem Constraint Optimization Problem [5] - Chapter 6 83

Late Acceptance Hill Climbing Late Acceptance Hill Climbing [17] 83

OST Eastern Switzerland University of Applied Sciences: In this document focused on the campus Rapperswil-Jona. 84

Regular Semester A regular semester is the semester where an exam normally takes place in. For example: The module “Objektorientiere Programmierung”, short OO, is in the first semester of your IT-studies. Therefore the regular semester of the module/exam OO is I1, which means “Informatik, 1. Semester” 4, 6

VisualVM VisualVM is a tool that provides a visual interface for viewing detailed information about Java applications while they are running on a Java Virtual Machine. [Wikipedia] 35

Acronyms

GUI Graphical User Interface 2

LAHC [Late Acceptance Hill Climbing] 24, 26, 38

REST Representational State Transfer 44

SA Simulated Annealing 23, 26

SCHC [Constraint Optimization Problem] 26, 38, 41

Abbreviations

C4 Context, Containers, Components, and Code (See: C4 model) 4, 85

OST Eastern Switzerland University of Applied Sciences (See: OST) 1, 61

List of Figures

List of Tables

List of Listings

Bibliography

[1] F. Germann and R. Jenni, “Automation of the OST-RJ Examination Scheduling,” OST – Eastern Switzerland University of Applied Sciences, Tech. Rep., 2020. [Online]. Available: https://eprints.ost.ch/id/eprint/917/

[2] E. C. Freuder, “In pursuit of the holy grail,” *Constraints*, vol. 2, no. 1, pp. 57–61, 1997. [Online]. Available: https://doi.org/10.1023/A:1009749006768

[3] W. Kohn, *Statistik*, 1st ed. Springer-Verlag Berlin Heidelberg, 2005.

[4] OptaPlanner, “OptaPlanner - Does OptaPlanner find the optimal solution?” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#doesPlannerFindTheOptimalSolution

[5] N. J. Nilsson, S. J. Russell, and P. Norvig, *Artificial intelligence: A modern approach (Third Edition)*, 2010, vol. 82, no. 1-2.

[6] OptaPlanner, “OptaPlanner - InitializingScoreTrend.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#initializingScoreTrend

[7] ——, “OptaPlanner - Planning entity difficulty.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#planningEntityDifficulty

[8] ——, “OptaPlanner - Construction heuristics.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#constructionHeuristics

[9] ——, “OptaPlanner - Scaling construction heuristics.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#scalingConstructionHeuristics

[10] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[11] OptaPlanner, “OptaPlanner - Accepted Count Limit.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#acceptedCountLimit

[12] ——, “Accepted count limit.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#acceptedCountLimit

[13] A. C. M. U. Platzer, “HillClimbing (Orbital),” 2009. [Online]. Available: http://www.cs.cmu.edu/~aplatzer/orbital/Orbital-doc/api/orbital/algorithm/template/HillClimbing.html

[14] F. Glover, “Tabu Search—Part I,” *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[15] ——, “Tabu Search—Part II,” *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[16] H. Pirim, E. Bayraktar, and B. Eksioglu, “Tabu Search: A Comparative Study,” *Tabu Search*, no. May, 2008.

[17] E. K. Burke and Y. Bykov, “The late acceptance Hill-Climbing heuristic,” *European Journal of Operational Research*, vol. 258, no. 1, pp. 70–78, 2017.

[18] Y. Bykov and S. Petrovic, “A Step Counting Hill Climbing Algorithm applied to University Examination Timetabling,” *Journal of Scheduling*, vol. 19, no. 4, pp. 479–492, 2016.

[19] L. Petrovický, “OptaPlanner: Is the “constraint match” associated with a score just a semantical thing?” [Online]. Available: https://stackoverflow.com/a/67816862

[20] OptaPlanner, “Avoid floating point numbers in score calculation.” [Online]. Available: https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html#avoidFloatingPointNumbersInScoreCalculation

[21] G. D. Smet, “Filter prevents assigning facts.” [Online]. Available: https://stackoverflow.com/a/67087489

[22] OptaPlanner, “Create a constraint configuration.” [Online]. Available: https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html#createAConstraintConfiguration

[23] ——, “optaplanner/defaultlocalsearchphasefactory.java,” https://github.com/kiegroup/optaplanner/blob/master/optaplanner-core/src/main/java/org/optaplanner/core/impl/localsearch/DefaultLocalSearchPhaseFactory.java#L177, (Accessed on 06/06/2021).

[24] ——, “Score constraint signum (positive or negative).” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/#scoreConstraintSignum

[25] ——, “Define the constraints and calculate the score.” [Online]. Available: https://docs.optaplanner.org/8.5.0.Final/optaplanner-docs/html_single/index.html#_define_the_constraints_and_calculate_the_score

[26] ——, “Score trap.” [Online]. Available: https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html#scoreTrap

[27] ——, “REPRODUCIBLE (default).” [Online]. Available: https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html#environmentModeReproducible

[28] ——, “Multithreaded Incremental Solving.” [Online]. Available: https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html#multithreadedIncrementalSolving