Maastricht University

DKE Scheduling Project
Department of Data Science and Knowledge Engineering

Anton Bulat, Fatimah Mulan Ahmed, Fred Shen, Maxime Laschet

June 22, 2016

Abstract

The aim of this work is to compute an optimal schedule for the students and teachers of the Department Data Science of Knowledge Engineering at Maastricht University. The research and the mathematically modeling of a solution take a central aspect of this project.
The work has split into two groups: The Integer Linear Programming and the heuristics group.
This report will focus on the work of the heuristics group. It will explain the conceptuation of the model including all constraints, the application of Local search and some heuristics approaches and experiments to improve feasible solutions.

1 Introduction

The modeling of a university course timetabling problem is a well known problem. There exists many commercial applications on the market. The fact that everyone applies some internal rules which are not implemented in the available versions transforms the simple problem into a challenging problem. For this project, the scheduling problem of the Department of Data Science and Knowledge Engineering has been chosen to investigate.
The goal of the autonomously running application is to produce a feasible schedule for different students, lecturers, which follow or teach different courses in different rooms during a given time. Several constraints are supposed to fulfill that a course can be scheduled in a given time slot. Once the

algorithm has found out all the different solutions, it exists the possibility to improve the calculated result. After the enhancement step the schedule will provide a smoother result which makes every involver more satisfied.

2 Problem Description

"Course timetabling as a multidimensional assignment problem in which students, teachers (or faculty members) are assigned to courses, course sections or classes; events (individual meetings between students and teachers) are assigned to classrooms and times." [2]
The DKE scheduling problem corresponds exactly to this simple definition. There are many courses given from one or more lecturers for a different set of students. The University disposes a given number of rooms with different capacities and equipment. All events should be scheduled between starting date and ending date. Each day in one period is split into four time slots:

- 8:30-10:30
- 11:00-13:00
- 13:30-15:30
- 16:00-18:00

So each week has 20 feasible time slots to schedule events.

The feasible solution of the schedule is defined as a solution which satisfies a list of rules, the hard constraints. The current scheduling problem involves the following constraints:

- A student only can follow one event per time point
- A lecturer only can teach one event per time point
- Each year of students has some days where there are not available
- A lecturer can be not available at a given time slot
- Only one event is allowed to be assigned to a time slot in each room.
- The room capacity should be equal to or greater than the number of students.
- The room assigned to the event should satisfy the features required by the event. Like a certain number of computers.

The feasible result can be improved by applying rules which are not necessary but enhance the quality of the schedule, which are soft constraints:

- A lecturer should not have too many theoretical classes on one day
- Students should not have a single course on one day.
- Students should not have long break between events.
- Students should not have many changes in room locations per day.
- Events for one course is supposed to be divided equally within all weeks.
- Equilibrium between theoretical and practical sessions of each course
- Some events have to be scheduled before other events
- Same courses in same rooms

The following sections explains how the problem has been modeled and which algorithmic solution was applied to calculate the academic schedule of the DKE.

3 ILP and Heuristics Group

The first analysis of the research project has been done with all students of the project. In the second phase, the students were split into two groups. One group is handling the problem with an Integer Linear Programming solution and the other group with an heuristics approximation to the solution. The following of the report explains the research and results of the heuristics approach.
The goal of the project also involves merging the results of both groups at the end. The results which are calculated by the Integer Linear Programming group is passed to the heuristics group and improved to achieve the final output for the schedule. A parser needs to be designed because both groups use different kinds of model structure and the data has to be translated from one model to another.

4 Application Design

The work was started by collecting all the different variables which involves the calculations of an academic schedule. Clearly, the data has to satisfy

all the different specifications of the courses, lecturers, students, rooms of the DKE. In the later phase the work will be merged with what is returned from the ILP group. Java was chosen as programming language. Compared with other programming languages, no significant differences exist for the heuristics method. The ILP group uses Gurobi solver, which is compatible with Java. [4]

The figures 4, 5 and 6 in the attachment show the UML class diagrams of the designed Java class model.

The first approximation was done over the different data which involved the calculations of the autonomous running application. Only after every detail in the model was available, the algorithm was written. So the structure was clear and it was possible that the procedure could handle all possible cases.

The "Timetable" contains the central information of the schedule. The starting date and ending date of the period are inputs for computing the needed number of time slots. For each academic year the "Timetable" stores a Hash Map of "Events", which allows a faster access because the keys contains the time slot number. Not all time slots contains an event and therefore the algorithm have to iterate the whole array when it is stored in an array. Each "Event" refers to a "Course", a List of "Rooms" and a hard-constraint-penalty for the event.
The "Course" dispose its characteristics like the internal id, the UM course id, the course name, a List of "Lecturer" ids, a List of "Student" ids. Furthermore, in both cases of ids list, the first element of the list is the principal teacher or student of the course. The number of theoretical and practical hours are saved as two integer numbers, additionally the course stores a Boolean variable stating if a projector and a whiteboard are needed for this course.

A List of the following classes is available in the application.

The "Student" properties includes an internal id, the name of the academic year, the number of students, a list of Days on which the students are available. As a last element, a list of days on which the project can be scheduled is stored. This field is explained in section 6.5 with more details.

The "Lecturer" contains an internal id, a name and an integer array of

availability. The size of the array is equal to the number of time slots.

The "Room" consists of the id, the name of the building, the room number, the number of chairs, the number of computers and the characteristic if it has equipment like a projector or screen and a whiteboard.

5 Algorithm

5.1 Local Search

"Local search is an iterative algorithm that moves from one solution S to another S' according to some neighborhood structure. " [7].

The local search is used to improve the schedule from last year step by step. Each event has a penalty value given by constraints in each time slot. The algorithm aims to find a schedule with the lowest possible penalty for all events assigned in the schedule. The process of the method is as follow:

- Start with an existing solution S. For example from the last year or an solution parsed from the ILP-application
- Examine penalty value for solutions S' in a neighborhood N(S) of S
- If the penalty value is minimized, which means improving the solution, accept the result
- Move to a good solution

$$S' \in N(S)$$

6 Application of the Core-algorithm

The algorithm runs for all academic years over all events to verify the feasibility with the hard constraints.
Afterwards, the algorithm continues conducting movements for soft constraints to improve the feasible schedule by minimizing penalty values returned by the soft constraints. In this case each soft constraint calculates its own value of violation, for each possible position of each event. If it minimizes the weighted sum of all soft constraints, the movement is preserved.

6.1 Hard Constraints

To check the feasibility of events on time slots, a hard constraints mapper is designed to iterate over all hard constraints and give True/False as return. Furthermore, it is necessary that every event satisfies all hard constraints in order to be included into a feasible schedule. All hard constraints within the algorithm is listed as below.
A more specific overview about the different constraints which are included in a feasible schedule.

- The availability for the different students
  - Bachelor year 1: Every weekday
  - Bachelor year 2: Monday, Tuesday, Friday
  - Bachelor year 3: Monday, Thursday, Friday
  - Master OR year 1: Monday, Tuesday
  - Master OR year 2: Every weekday
  - Master AI year 1: Thursday, Friday
  - Master AI year 2: Every weekday
- The lecturer availability is a property of the object and are read through individually
- Room properties are verified in relation to the given course and related number of students. Also the information of theoretical and practical hours
- Room overlapping is verified by investigating current booked rooms, for a given time slot

To carry out local change over an input schedule, a set of local movements are designed. The algorithm starts iterating over movements on events and verifies with hard constraints to generate a feasible schedule. Unlike the soft constraint case, the application just attributes if the event can be scheduled at the time slot or not. If all the hard constraints are violated, the algorithm marks the event with a value “PROBLEM”. This marker allows user to see whether all events are placed in a feasible time slot. If no hard constraint is violated, it is marked as “GOOD”.
It is to mention that in the general case the input is a feasible schedule from the last year or the feasible output of the ILP-application. If there are no big differences between the courses and lecturer availability from the last

to the current input, then the movements are not executed. If the input solution is feasible, all events are checked, then it is marked as "GOOD". The application will continue with the soft constraints part.
In section 2 the hard constraints are explained already in detail. For each case a Boolean verification function is called. It checks the individual properties of the rule in context to the given year and event in a specific time slot. If one of them returns false, the event doesn't satisfies all conditions and is therefore not feasible.

6.2 Soft Constraints

Next to iterating over hard constraints, the feasible schedule will be improved by iterating over soft constraints. Soft constraints will have more relaxing criteria for validation and return integers as penalty values. The goal is to minimize the total weighted sum of the soft constraints.
The algorithm 1 gives an overview over the procedure. Each provided constraint will be checked and sum up to a general penalty value. The weight for each constraint was attributed in the testing phase. It allows to attribute a higher or lower importance to different constraints. A formula is expressed as below:

$$Minimize : \omega(1) * p(1) + \omega(2) * p(2) + ... + \omega(n) * p(n)$$

where

$$n$$

is the number of constraints,

$$\omega(n)$$

is the weight assigned to each constraint and

$$p(n)$$

is the penalty value returned by each constraint.

The first part explains how to calculate or attribute the penalty value. The section 6.4 explains which movements can be done to improve the schedule. Lower penalty values indicates the improvements of the schedule.

6.2.1 Day Utilization

The first constraint subtracts a sub schedule for each day and for each academic year. Together with checking whether each time slot is occupied

Algorithm 1 Get weighted penalty value for all soft constraints

```
procedure GET WEIGHTED PENALTY VALUE FOR ALL SOFT CONSTRAINTS
    init:
    counter ← 0
    numberOfSoftConstraints ← softconstraints.size
    penaltyValue ← 0
    top:
    for counter = 0; counter < numberOfSoftConstraints; counter++ do
        constraintName ← softconstraints.get(counter).getName()
        constraintWeight ← softconstraints.get(counter).getWeight()
        penaltyValue = penaltyValue +
 getPenaltyValueForSoftConstraint(constraintName) * constraintWeight
    end for
    penaltyValue
end procedure
```

or not, all situations of a day sub schedule can be generated. Moreover, several levels of penalty values are decided for distinguishing all situations. The testing criteria of the first soft constraint is shown below:

1. Only one time slot is occupied, which receive the highest penalty value 3.
2. Two time slots in the middle are empty, which means students will experience long waiting time. A penalty value of 2 is given.
3. Situations where the gap is only one time slot also where four time slots are occupied, then a penalty value of 1 is given.
4. Situations where two or three time slots are continuously occupied, a penalty value of 0 is given.

By minimizing the penalty value returned for each period, the gaps between time slots can be reduced.

6.2.2 Equilibrium

For the second soft constraint, the aim is to allocate events as evenly as possible among all weeks. The method is to record the frequency of occurrence of a selected course also the desired frequency of that course. A

penalty value is given if a mismatch of these two frequencies are detected. The algorithm minimizes the penalty value returned from the constraint.

6.2.3 Change Building

Similar as the first constraint, the third soft constraint also applies on sub schedule under same conditions. Rather than checking about the sparsity of the sub schedules, the frequency of changing lecture rooms among different buildings is checked. For A a day sub schedule, once the next time slot is assigned to a lecture room in a different building, the penalty value is increased by 1. And the penalty value is initiated with 0. After looping through each period, the penalty value is minimized to ensure a low frequency of changing rooms.

6.2.4 Full Lecture Day

The fourth soft constraint focuses on teacher availability. The whole list of lecturer is looped through and for each lecturer, the availability per day is checked. If a lecturer has more than three lectures per day, a penalty value 1 is given then for each day. By applying the process, the algorithm attempts to avoid fours events in one day for a lecturer.

6.2.5 Theory before Practical

The last soft constraint checked the arrangement of theoretical and practical courses of a selected course. For each selected course, the algorithm first count for each week how many theoretical course and practical course are assigned. And in total eight situation are included inside the algorithm.

1. only practical courses exist, penalty value 3 is given
2. only theoretical courses exist, penalty value 0 is given
3. when the first course of the week is theoretical and the amount of theoretical course and practical courses equals to each other, penalty value 0 is given
4. when the first course of the week is practical and the amount of theoretical course and practical courses equals to each other, penalty value 2 is given

5. when the first course of the week is theoretical and the multiple of theoretical course and practical courses equals to 2, penalty value 1 is given

6. when the first course of the week is practical and the multiple of theoretical course and practical courses equals to 2, penalty value 2 is given

7. when the first course of the week is theoretical and the multiple of theoretical course and practical courses is larger than 2, penalty value 2 is given

8. when the first course of the week is practical and the multiple of theoretical course and practical courses is larger than 2, penalty value 3 is given

Depending on these checking conditions, a penalty value of each week is computed and the accumulation is returned for the minimizing function.

Finally the algorithm profits from the fact that the input is already a desired schedule. So practicals are already scheduled in practical rooms. Same courses are scheduled each week in the same room and more or less at the same time slot.
The experiments on the scheduling improvement have approved that these good starting conditions helps significantly on the algorithm.

6.3 Movement for Hard Constraints

Several actions try to improve the feasibility of each event which has been marked as not feasible at this time point. The order of the actions has been decided by test and logically argumentation in relation to the soft constraints. In the following cases the only goal is to find a time point where the event can take place. It can be a good or worse moment in relation to soft constraints.

6.3.1 Find New Room

To start with, the algorithm attempts to find an available room for a selected event, by which it causes least change to the schedule. Before finding a new room for a selected event, the lecturer availability is checked at the first place. If the lecturer is available, a search for free rooms at the time slot will continue by looping through the room list. Once an available room is

found, a test for hard constraints is conducted. The movement is accepted when no hard constraint is violated, otherwise the selected event is rolled back to the original time slot.

6.3.2 Move to an Empty Time Slot

The second movement is to search for a new free time slot for a selected event. The algorithm starts with an event on every Monday and then loops through the corresponding week schedule to find an empty time slot. The Monday has been chosen to keep directly an equilibrium of the entire schedule. If the counter starts at the beginning, the schedule will have many events at the beginning and few at the end, vice versa. So it moves the selected event to a new time slot and update the availability of lecturer. Furthermore, a test on hard constraints is conducted to validate whether the new movement violate the hard constraints. The new movement is only kept when the hard constraints test is passed, otherwise the movement of the selected event is rolled back to the original time slot.

6.3.3 Move with an Existing Event

The third movement swaps two events to reduce the violation on hard constraints. Similar as the previous movement, the algorithm picks an event on every Monday and loops through the corresponding week schedule to find an event to swap. After swapping the time slots of two events and updating the availability of lecturer, hard constraints test is applied.

6.3.4 Main Lecturer Availability

The last movement checks whether the main lecturer of a selected event is available. In the situation where not all lecturers of a selected event are available, the algorithm attempts to replace the list of lecturers with the main lecturer. Before the replacement is carried out, the availability of the main lecturer is verified. If the main lecturer is available, the replacement is continued.

To sum up, the four movements above perform local changes on events and produces a feasible schedule by validating with hard constraints. The feasible schedule is further processed by conducting movements and minimizing penalty value of soft constraints. The total penalty value is a linear combination of penalty values returned from each soft constraint, where co-

efficients are individual weights assigned to each soft constraint. The process of movement with soft constraints is described in next part.

6.4 Movement for Soft Constraints

Three cases of movements are included in movements for soft constraints.

6.4.1 Move to an Empty Time Slot

The first case is similar to moving an event to an empty time slot as described in the case of hard constraints with an adaption of checking soft constraints. After moving an event and verified with hard constraint test, the total penalty value of soft constraints is calculated and compared with an initial penalty value. The goal is to minimize the total penalty value, so the event is only moved when the calculated penalty value is lower than the initial penalty value. On the other hand, if the event is not moved, the algorithm continues with next movement.

6.4.2 Move with an Existing Event

The second case of movement for soft constraints is adapted from swapping two events in the case of hard constraint movement. By verifying with hard constraint after swapping two event, the total penalty value is calculated and compared as described in the previous case. The swapping is carried out only when the total penalty value is reduced. If the total penalty value is not reduced, the algorithm continuous with the third case.

6.4.3 Find New Room

The third case of movement for soft constraints is to find a new room. If a new room is available without violating hard constraints, the same check as in previous cases is conducted to ensure the movement is done only when the total penalty value is reduced.

6.5 Pre-processing

The Local search algorithm starts with an existing solution as mentioned. This solution can be the schedule from last year. Even a manually adapted version or the output from the ILP-Application can be taken as input. Before the algorithm starts, it is always a good practice to verify the input. The pre-processing focuses on two parts of the projects and the new or old courses part.

6.5.1 Project

The students from the bachelor years can follow either a project or a DKE@Work program. In the constraint list of the bachelor years, it was already mentioned that two given days of the week have to be blocked for these students. In consequence, several students work on these days in different companies, while the remaining students should work on these days on their projects. This means that a project meeting should be placed on a day which is marked as not available for the courses. When the algorithm tries to verify the hard constraints it indicates that it is not feasible on this day.
Therefore all project-events are verified in a pre-processing iteration. All events are iterated for all years and are fixed if the algorithm has found a feasible position. This placement implements the knowledge of a supplement parameter, the project day for each year.
Since the project events are fixed, the main algorithm will not move them.

6.5.2 New or Old Course

The algorithm only moves events from one time slot to another time slot. So what happens if there is a new course which is not in the solution from last year? Furthermore, a course can have more or less events in the upcoming year. The following section will explain the different pre-processing steps such that the input contains the right data.

Non Existing Courses

Before the algorithm can start the movements it has to verify that there are not new courses given in this period. It will check for all courses if there are at least one occurrence in the corresponding year. This verification is done by id comparison. If there is not, it will schedule the number of events in the next free, available time slots. The number of events is defined in the course class. So a new course can be added in future schedules without having it in the feasible solution from the last year. In opposite to this case is when a course can no longer be given. In this case all events are simply removed from the schedule. The lecturer availability have to be updated.

Number of Events in the Given Solution

It is possible that the number of theoretical or practical events has changed from one year to another year. The pre-processing function counts all events for each course. If a course has too many occurrences it will delete the last

events. If the schedule contains less events than indicated number for the event, the application autonomously add an event for the course to an free, feasible time slot.

6.5.3 Lecturer

The lecturer availability was first modeled with a Boolean array. The problem occurred by applying the hard constraint, which checks the lecturers' availability of the given time point. Since the lecturer was teaching a course in one year, his availability was false. So that he could not teach another course at this moment. In this modeling there was no reference between course and availability. The check of the hard constraints at the time point for the event will not be satisfied because the teacher is not available at the moment.
Furthermore there was no reference to a non-availability in case of non DKE events. The lecturer can become available by moving the corresponding DKE event to another time slot. So the availability sores a number corresponding to year in which he teaches and a specific value for never-available. Additionally in the pre-processing step the given availability is compared and if needed modified to the given input schedule.

7 ILP Output as Input

The Integer Linear Programming group creates their own solution which implements different model structure from what the heuristics group's application uses. The heuristics group is supposed to work with this solution and improve it. To use the calculated data, ILP's output is translated into an input for heuristics application. A parser creates a JSON file based on the data from the output. The JSON file has the same structure that is used for the scheduling without any precast solution. The details of the parser are discussed in the next section.

7.1 Parser Description

The output from the ILP-application is a text file which contains all the different events in different rows. One event consists of a course-id, lecturer-id and a room-id, which is very similar to the input of the heuristic application. Nevertheless there are some drawbacks. The internal used id numbers are

not the same. So the id number for a room, lecturer, and course are different in the two models. The ILP-application knows only the time slots from Monday to Friday, however, the heuristics also includes Saturday and Sunday. What's more, Virtual rooms, which means that a course in one time slot uses more than one room is only inclusive in the ILP group. While the same problem is solved with a list of rooms in the heuristic application. The holidays and fixed events are hard coded by the ILP application, which is a problem since the heuristics group creates these events and block the corresponding time slots. Furthermore, no events can be fixed to a time slot or read out in the ILP-application, which means that they cannot move. Based on all the several special cases a parser was designed. The parser distinguishes the known problems between the two applications. At the end an input file in the JSON format is generated. The file can be imported by the heuristic application and handled as a normal input.

8 Utilization of the Application

The application starts with reading in all the files which are in the "data/input" folder. These files are JSON (JavaScript Object Notation) files. The Java objects of the different classes are serialized in a specific format to specific text files. The import and export uses functions of the Jackson [3] library. This approach allows to store easily the plain old java objects to a persistent format. The output of this year can be taken as input of next year. Simply by moving the output files into the input folder.

The text files are readable for software engineers which are up to date of the project's structure. The normal end user will have difficulties to understand the structure. Therefore a simple user interface was built to show how the application can be adapted over layer.

So the application has read in all the different variables over the persistent files. Then the data is visualized in tables with tabs corresponding to the category. The figure 2 shows the structure of the user interface. Here the user has the possibility to create, read, update and delete the different entries, which enables users to add a new course, update the availability of a lecturer, modify the size of students of an academic year and last but not least the possibility of manual configuration of the schedule figure 3.

All scheduled events from the existing solution are visualized in the table.

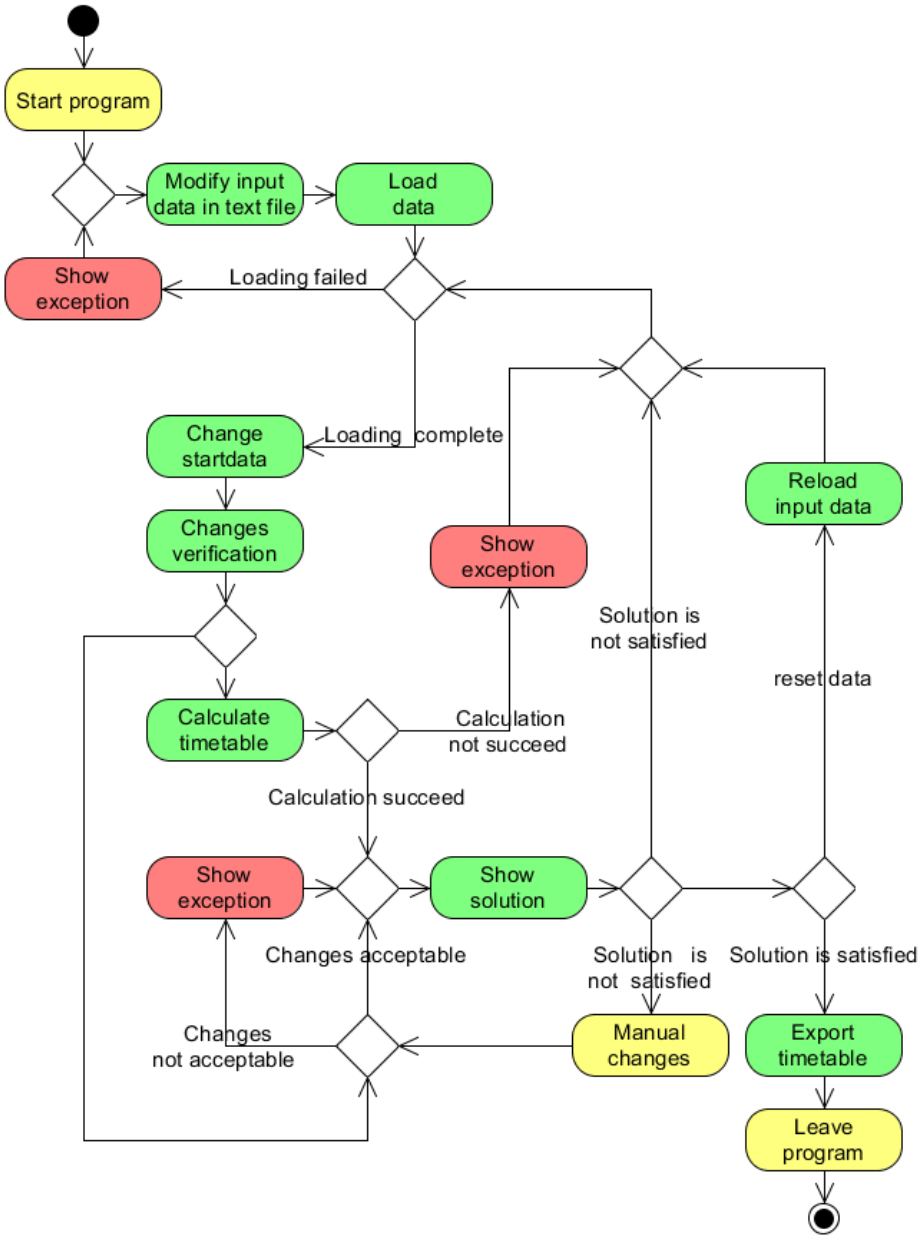

Figure 1: Activity diagram

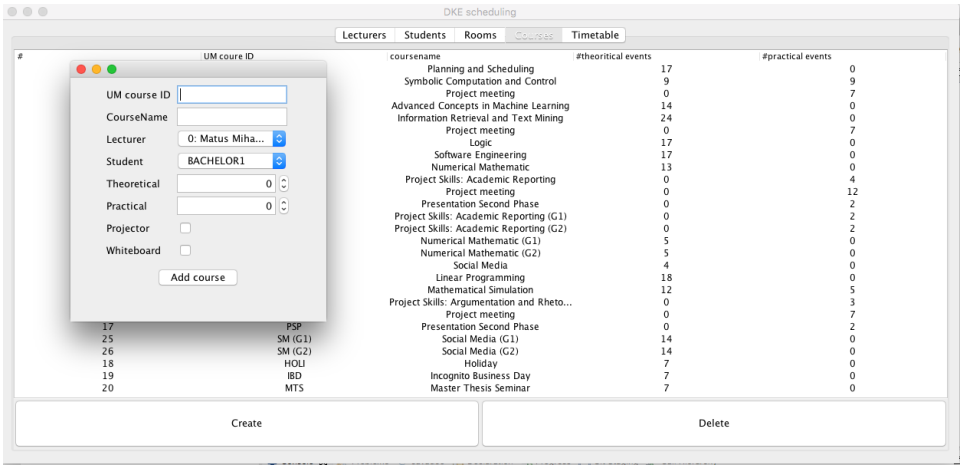

Figure 2: User interface: Overview courses

The user has the possibility to fix one specific event, to move a specific event to another time slot and to change a room of a selected event. The movement and the attribution of a new room over the user interface is bounded with the hard constraint verification, whenever it is possible (only for the user, not for the algorithm) to move an already fixed event. The algorithm can be started once the user is satisfied with the modifications. The table will be updated with the new calculated data from the algorithm which applies the hard and soft constraints. Until the proposed solution is not satisfied, the user can apply his manual changes followed by executing the algorithm or exporting the schedule.

Once this process is done the Application exports automatically the schedule in three different formats. The first is the JSON file to be used later as input, the second is the schedule for this period as HTML file, which is possible to be included in a web page and be saved as PDF. The last corresponds to the layout of the current version which knows the students of the DKE. The translation from HTML to PDF has been done with the iText library [5].
If the user is a software programmer or can read the structure of the JSON files, then there is no difference between the graphical input and the text file based input.

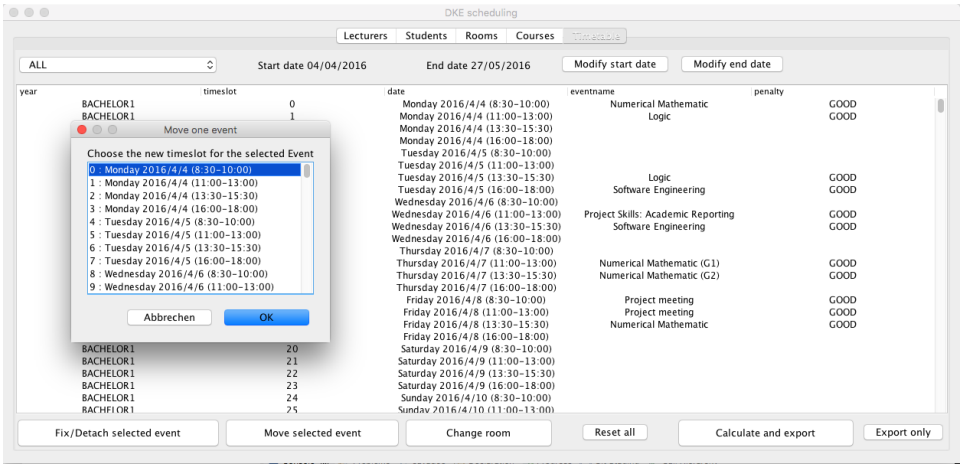

Figure 3: User interface: Overview schedule

9 Performance of the Application

The number of iterations for the algorithm depends highly on the given input. Either a lot of changes are applied or the most part remains the same. In the best case it runs only once over all the events. In the general case, for a given feasible solution, the application calculates about 5 seconds. If the provided input is not feasible and a lot of changes are involved, the application finish after 15 seconds.
Additionally, the following run-times estimates the worst case, however, the possibility of achieving this case is really small. The experiments with different inputs have shown that a solution could be found after a few steps. The hard constraints check will be done like the most cases in $\mathcal{O}(year*timeslots)$, for iterating all time slots for all years.

10 Conclusion

Starting from a given schedule, the algorithm first verifies its feasibility. After movements of the hard constraints, a feasible schedule is achieved. Next a local search with soft constraints is applied to improve the returned solution. Therefore, a feasible and improved schedule is created for the DKE scheduling problem.

Furthermore, a graphical user interface allows to inspect the calculated

solution. Freedom of manually changes and reiterating over the algorithm was designed as features to satisfy user's requirements.

11 Future Plan

For future plan, expanding the mathematical model is the first step. New constraints can be modeled and added to the algorithm, as an enhancement for the quality of schedule. Moreover, an investigation of the performance of weights and penalty value is to be conducted. In details, different combinations of weights and penalty values will be examined and optimized according to results. Merging between output of ILP group and input for heuristics group is also part of the future plan. Last but not the least, a function of detecting and fixing holidays automatically should be created.

References

[1] H. Arntzen and A. Løkketangen. A local search heuristic for a university timetabling problem. *nine*, 1(T2):T45, 2003. [Online; accessed 19-June-2016].

[2] M. W. Carter and G. Laporte. *Recent developments in practical course timetabling. In Burke, E., and Carter, M., eds., Practice and Theory of Automated Timetabling II, 3–19.* Springer-Verlag, 1998.

[3] Github. Jackson project home @github. `https://github.com/FasterXML/jackson`. [Online; accessed 18-June-2016].

[4] Gurobi. Gurobi-java api overview. `http://www.gurobi.com/documentation/6.5/refman/java_api_overview.html`. [Online; accessed 18-June-2016].

[5] iText. Superior pdf creation and conversion. `http://itextpdf.com`. [Online; accessed 18-June-2016].

[6] T. Müller, K. Murray, and S. Schluttenhofer. University course timetabling & student sectioning system. *Space Management and Academic Scheduling, Purdue University*, 2007. [Online; accessed 18-June-2016].

[7] K. Ponnalagu, R. S. Rajan, and B. Sengupta. Automatically generating high quality soa design from business process maps based on specified quality goals, Sept. 20 2010. US Patent App. 12/885,870; [Online; accessed 18-June-2016].

[8] O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, K. Socha, and B. Paechter. A local search for the timetabling problem. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, PATAT*, pages 124–127, 2002. [Online; accessed 19-June-2016].

12 Appendix

UML Class Diagram

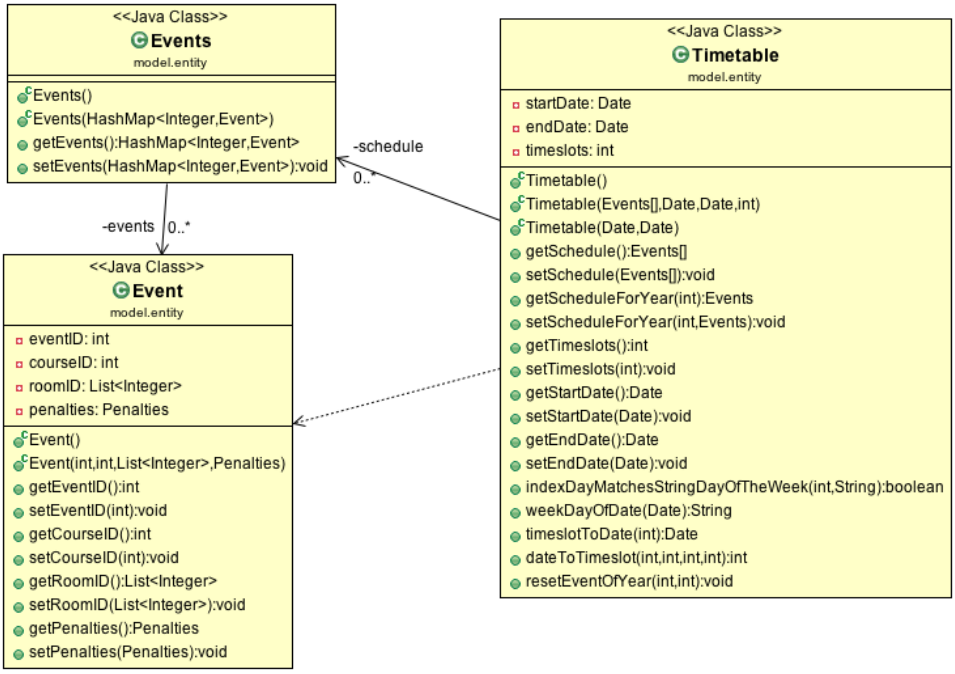

Figure 4: Class diagram: schedule (1)

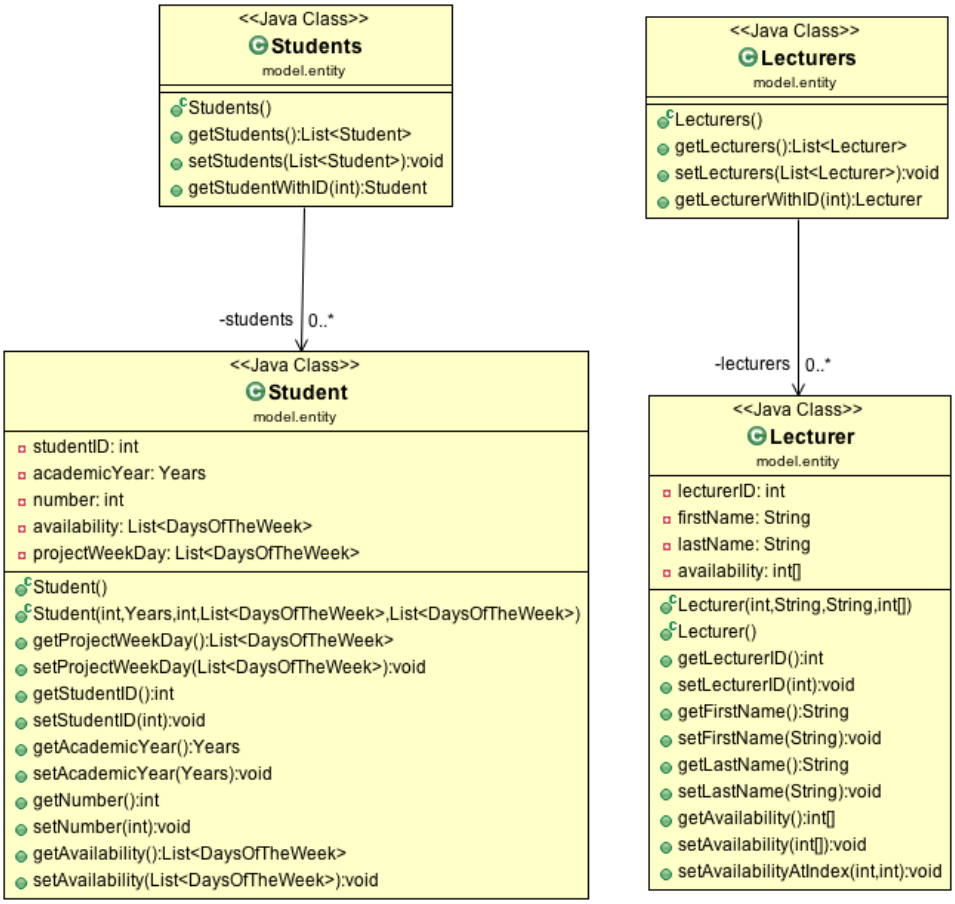

Figure 5: Class diagram: schedule (2)

Figure 6: Class diagram: schedule (3)

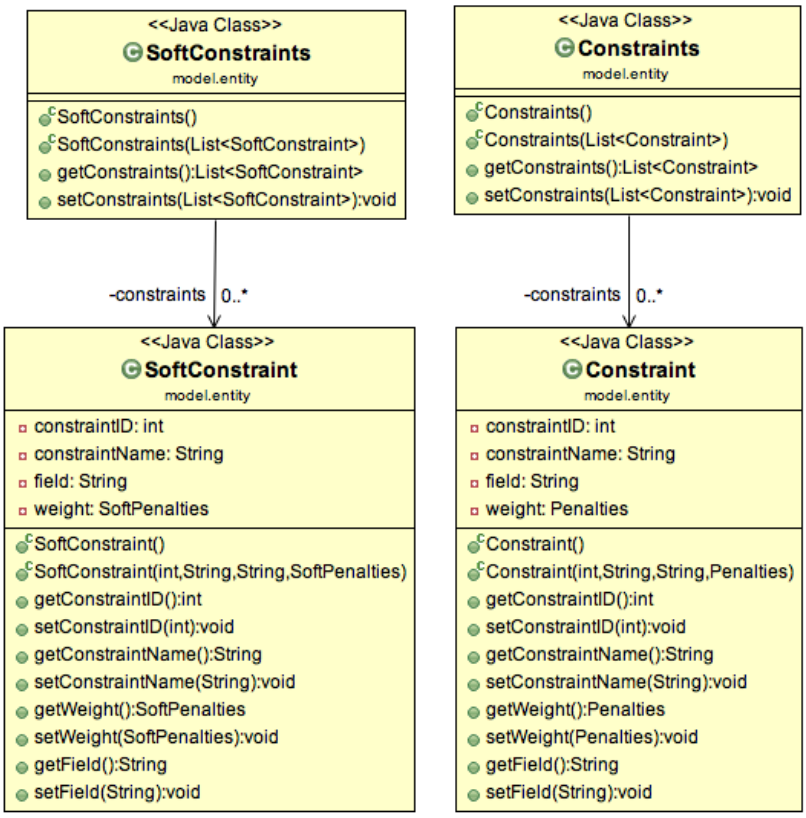

Figure 7: Class diagram: constraints

A Hand Out of How to Use the Implementation

The project is furnished as an existing project in a Java work-space. The structure is designed as model-view-controller. The data entities and the algorithm are stored in the model. This functions are called by the controller and the answer can be shown in a view, also handled by the controller. The section **??** describes in detail the application work flow. The code is commented and all functions have a logically name in context what they are doing.
The utils package contains some helper functions like translations, enumerations or import export features. The UML class diagrams of the different entities are stored in the entity folder, so that every software developer can have a quick model overview.
The algorithm class contains the core-algorithm. The main part is the "startTheTestForTheSchedule" function that verify all events in the schedule and the "moveEvent" function which is called by the last function.

The Format of the Input and Output

The input format of all variables can be done by adding values in the different JSON files in the data/input folder. This input corresponds to the entities of java classes, serialized objects in a persistent JSON file. See as an example for the input of Period 5, the "GenerateTestData" class. This class create all variables as java objects and export its to the text file of the related object.
For the scheduling algorithm the application read in this file in objects and trade during the process only the Java object, not the file. As software developer you are able to modify these objects during the run-time. At the end the modified data is serialized to a new JSON file in the data/output folder.
As supplement a HTML file which has the know layout of the DKE schedule is exported after each iteration. A non modifiable copy is generated as PDF file. So there exists an output for the machine, which will be used in a next iteration by the application and a readable output for the end-user, the students.

Documentation of How to Incorporate Improvements into the Code

As indicated, the application is developed in the model-view-controller design. The controller handles the actions between the model and the view.

The model is able to calculate or trade the data without the view. The main part of the application is the Algorithm class, from which the data is read in. The input data is verified and then the feasibility and improvement steps are executed. To improve the feasibility, the modifications should be done in the HardConstraint class, for a smoother schedule the calculations and movements are done in the SoftConstraint class. To add parameters to the data model, the entities can be found in the model.entity package. The application works every time on the data which is stored in the Algorithm class attributes. The data used is always a reference to the original data. This is an important note for the hard constraint check. The checks are done on the real data, not on duplicates given as parameter.

A list of Recommendations for Improving the Given Implementation

Handling of the special events can be improved. Actually the algorithm works well for all known special cases. It is possible that the algorithm has to be changed, if a new case is added in a later stage.
One of the special events are the holidays. In the current version they are added manually and changes are only done independently of the different years. So the possibility should be added in order to move a set of related events to a new time point. Additionally to detect the holiday events from a calendar automatically.
The graphical user interface was created to give the end-user a quick view over the application. The main functionality are available over the interface. In the application behind all scenarios are possible with the given structure, so if something is missing it should be added to the view and bounded to the code.