Maastricht University

Department of Data Science and Knowledge Engineering

Master's Program in Artificial Intelligence
and Data Science for Decision Making

Master Research Project - Report

Group 18

June 25th, 2020

Timetable Scheduling for the Department of Data Science and Knowledge Engineering

Authors:
Yu Fee Chan
Paul Disbechl
Daniel Kaestner
Camilla Lummerzheim
Le Duc Huy Ngo
Guillermo Quintana Pelayo

Supervisor:
Dr. Matúš Mihalák

Coordinators:
Dr. Kurt Driessens
Dr. Gijs Schoenmakers

Abstract

Timetable construction is an actively studied field of research. Integer Linear Programming (ILP) and metaheuristics are regarded as most successful for automated timetable generation. In this paper, we investigate whether these approaches could be applied to generate timetables for the Department of Data Science and Knowledge Engineering (DKE) at Maastricht University. Further, we investigate whether these schedules can also be improved in terms of student satisfiability. Lastly, we aim to incorporate this in an application which can be used by the department scheduler to create optimal timetables.

Keywords: *automated timetable generation, university timetabling, Integer Linear Programming, metaheuristic*

Acknowledgements

We would like to thank Dénise van Spingelen (Schedule Planner at DKE) for sharing the ins and outs of real-world timetable scheduling throughout our research.

Contents

1 Introduction **4**

2 Definitions **4**

3 Context and Motivation **6**

4 Previous Work **7**
- 4.1 Creating Qualitative Timetables 8

5 Schedule Evaluation **9**
- 5.1 Survey Design and Results 9
- 5.2 Evaluation Score Implementation 11

6 Approach / Algorithms **12**
- 6.1 Framework 12
- 6.2 Baseline Greedy Algorithms 13
- 6.3 Metaheuristics 14
  - 6.3.1 Tabu Search 14
  - 6.3.2 Genetic Algorithm 15
  - 6.3.3 Memetic Algorithm 17
- 6.4 Integer Linear Programming 18
  - 6.4.1 Notation 18
  - 6.4.2 Mathematical formulation of the ILP 20
  - 6.4.3 Complexity and Analysis 23
- 6.5 Usability 24

7 Results **26**

8 Conclusion **27**

9 Future work **28**

1 Introduction

University schedules are structured plans for students and staff members indicating the time, date and location of classes and events. Due to the sheer number of variables involved, constructing a university schedule is a complex task, since these variable impose limits on how events can be scheduled. These limiting variables are called constraints. Currently, at Maastricht University's (*UM*) Department of Data Science and Knowledge Engineering (*DKE*), schedules are painstakingly made by hand for almost 60 courses per year with many constraints, such as room and staff availability. In order to alleviate the workload of the scheduler, a program has been designed which aims to generate optimal prototype schedules which are then finalized by the scheduler at DKE.

A set of rules exists which the schedule must adhere to is known as constraints. An example of a constraint no two classes can be given at the same time if they are in the same room. The set of schedules which do not violate such rules may be large. However, satisfying rules that should not be violated does not imply optimality, since this depends on personal preferences. These individual preferences can be used as a measure for the quality of a schedule.

In this report, terms and definitions used throughout the paper are first clarified in Section 2 to avoid misunderstandings of the reader. Section 3 provides additional information about the importance and impact of this project along with the project goals and research questions. Following this, Section 4 provides an overview of the state of the art and previous work and existing algorithms that can be used to build schedules. In Section 5 we elaborate on the survey that has been performed to learn the preferences of schedules from a student's perspective which were used to evaluate the algorithms. In Section 6 the algorithms which were developed to construct schedules are investigated. Section 7 compares the performance of the algorithms on a limited testing environment. Section 8 reflects a conclusion about the whole project and is followed by potential future work which can be investigated in Section 9.

2 Definitions

To avoid misunderstandings, it is important that the reader is familiar with the terms used throughout this report. Therefore, this section seeks to clarify the terms and definitions which are used:

1. *Period*

   A period is a block of a certain number of weeks. At DKE, a period has a length of 8 weeks.
2. *Schedule*

   A schedule is a plan for a period, of events which need to take place at a certain location and timeslot. In the context of a university schedule, a schedule may also be referred to as a timetable. These terms are used interchangeably in the report.
3. *Event*

   The events usually represent lectures or practical sessions taught by a staff member to students who are enrolled in a particular course. Such events have a duration of two hours. Additionally, events such as staff meetings or gatherings such as celebrations or open days might be planned.
4. *Timeslot*

   Events are allocated to certain timeslots. In this project, timeslots are considered to have a length of two hours, starting from 08:30, 11:00, 13:30, or 16:00.
5. *Classroom*

   The location of a lecture event is referred to as a classroom.
6. *Year group / Programme*

   A year group or programme is a group of students that is enrolled in a certain education year. At DKE, five year groups are present that follow courses: BAY1, BAY2, BAY3, MAAIY1, and MADSDMY1.
7. *Lecturer*

   Lecturers are the staff members that need to be present during a lecture event.
8. *Hard constraint*

   A schedule must satisfy a set of hard constraints which are rules that cannot be violated. Most of these rules occur very naturally and deal with overlap. A lecturer cannot teach two courses at the same time. Likewise, a student cannot attend two lectures at the same time and two lectures cannot be scheduled in the same room at the same time. Another hard constraint is that sometimes lecturers are not available. This could be because a lecturer attends a conference or because the lecturer of a class is from a different university and can only give lecturers on specific days. A schedule which violates a hard constraint is infeasible.
9. *Soft constraint*

   The set of feasible schedules is usually large. To differentiate the quality of feasible schedules, soft constraints may be used, which are typically based on individual preferences. For example, some people may prefer lectures which take place early in the morning whereas others may prefer lectures in the afternoon. A schedule which violates soft constraints can still be feasible.

10. *Quokka*
    A small furry furball creature native to Western Australia. They are well known for their schedules, even when quarantined.
11. *Lemur*
    See *Madagascar* (2005)

3 Context and Motivation

Generating feasible university schedules is a challenging problem, both due to the inherent complexity of the problem and the large amount of data that is required for real world verification of the result [1]. Additionally, since almost all variables involved are typically subject to change, university schedules have to be revised frequently. Since creating and revising these schedules is a tedious task to do by hand, it is investigated whether an automated, algorithmic approach can alleviate some of the work required. Since the optimality of a schedule depends on the individual preferences of the users of the schedule, an automatically generated schedule which satisfies the majority might not be optimal. Thus, the task is to create software which generates a feasible, (near) optimal schedule which at least satisfies the given hard constraints. Should the generated schedule be slightly sub-optimal from the perspective of the department's scheduler, the generated schedule should allow for small changes to be made by hand so that optimality can be achieved.

Research questions Given the context and the problems defined, this leads to the following research questions:

- How can the current schedules at DKE be improved in terms of student satisfaction and construction time?
- How can one choose which soft constraints are most important at DKE?
- Which features of a schedule are important to students and how should the constraints be weighted?

- What is a good evaluation metric for comparing timetables, and how should the cost functions be defined?
- How do the different algorithms compare in terms of their runtime and regarding the quality of the schedules they generate?

4 Previous Work

The problem of creating feasible university timetables can be described as the task of assigning events (lectures, meetings, etc.) to a set of time slots of the resources (lecture rooms) whilst satisfying all feasibility constraints. With a nominal amount of studens, courses, staff members and locations, this task is fairly complex and time-consuming. Cooper and Kingston have shown that the timetabling problem belongs to a family of NP-complete problems [2]: Taking different well-known NP-complete problems, they use polynomial bounded transformations to reduce them to timetabling problem variants. The NP-completeness means that the nature of the problem instance has a large impact on the runtime when creating a reasonable good timetable.

Some studies have shown that [3, 4] metaheuristic approaches such as memetic, evolutionary, genetic, tabu search... as well as IP/LP (Integer programming/Linear programming) methods are very popular for creating solutions to this type of scheduling problem.

Over the years, various automated timetabling approaches have been proposed. Because of the nature of this problem, most of these approaches consider the use of approximation algorithms, which try to get as close as possible to the optimal solution using a small runtime. Specifically, there has been an increase in the use of metaheuristics [5, 6, 7], which are essentially algorithmic frameworks that can be adapted to be suitable for different optimization problems. Next to these, in more exact approaches, the problem is formulated as an ILP, which is then solved using solvers such as Gurobi, CPLEX, or Python MIP.

In a survey performed by Lewis [8], the author concentrates on the five main paradigms of metaheuristics: Ant Colony Optimization, Evolutionary Algorithms, Local Search, Tabu Search, and Simulated Annealing. This survey focuses on the different ways in which these algorithms can be used to distinguish and handle certain hard and soft constraints. The algorithms are separated into three categories: one-stage optimization, two-stage optimization, and algorithms that allow for constraint relaxation.

It is shown that certain approaches have a better performance depending on the user requirements, and that there is no universally superior algorithm. For this reason, we aim to find approaches that suit the scheduling of DKE courses the best.

Regarding ILP, Daskalaki et al. present a $0-1$ ILP formulation in a case study for the timetabling problem, as seen in many universities [9]. They show that solvable and flexible models can automatize the timetabling process. In another paper, Daskalaki et al. present a two-stage relaxation approach [10]. In the first stage, some constraints are relaxed, which are then recovered in the second stage. They show that this approach saves computation time, while maintaining the solution quality. Furthermore, their approach allows for certain interactions with the users during the construction of the timetables. Wright [11] tried to improve the routine-ness of a timetable by trying to schedule courses for a single week and then repeating it, resulting in a faster but possibly infeasible solution compared to scheduling an entire period at once. This approach used as the basis for one of the baseline greedy algorithms, as an adaptation of and alternative to the regular greedy algorithm, described in Section 6.2.

In the past, another group of students attempted to automatic create DKE timetables [12] in a somewhat similar fashion. Specifically, these students investigated the use of an ILP approach and a heuristic-based approach. Since the present project needs to rely on specified constraints and input data, starting a new framework to work with is less time consuming than adapting the ones in that project to our objectives. It also gives us freedom to try new implementations and solve in a different way the problems they also encountered.

4.1 Creating Qualitative Timetables

Whether a schedule is feasible or not is fairly clear-cut: it either satisfies the hard constraints it is subjected to or it does not. Whether a schedule is optimal, however, depends on the definition of optimality. The risk of increasing constraints in the problem is that this can limit the number of feasible solutions to the point that no feasible solution exists for the constraints provided, for example in case of overlap. Since it is possible that constraints which the schedule must satisfy conflict with each other, algorithms used to solve the problem may have to allow for some slack; this eventually allows constraints to be violated. In a real-world application though, feasibility constraints cannot be violated.

5 Schedule Evaluation

To evaluate whether a schedule is of adequate quality, several tasks need to be carried out. One has to assess the results, compare the different algorithms, and determine the fitness of the metaheuristic search methods that were implemented. The quality of a schedule is determined whether and to what degree constraints are satisfied. A distinction is made between hard constraints and soft constraints.

Hard constraints must be satisfied for a schedule to be feasible. Not satisfying a hard constraint is therefore an undesirable option, where an algorithm must apply a large penalty or not allow the creation of these schedules at all. Hard constraints all take the form of binary decisions - students, lecturers and rooms are each either available or unavailable at each timeslot. Thus, these can be modeled as binary decision variables; they are either satisfied or violated.

For the soft constraints on the other hand, a lot more variety is possible. For this project, the goal was to base them on the preferences of the students, while the hard constraints are given through conditions of the department. Soft constraints can be modeled as decimal values representing the percentages of students who voted for a particular option. Further, the priority of each soft constraint can be taken into account based on how important students found certain preferences. In all, we model the problem as a sum of costs to be minimized (or respectively a sum of rewards which are to be maximized), in order to maximize the quality of the schedule.

5.1 Survey Design and Results

To identify these soft constraints, all current DKE students were invited to participate in a survey. This survey contained questions about different characteristics of a timetable and asked students to rate these or give their preferences. It was set up together with the scheduler at DKE to make sure the questions and response options were designed in a way that made the results usable. For the maximum hours of lectures per day, the answer "2" does not make sense as it is impossible to place all lectures then. Besides, it also asked general questions about student satisfaction with the schedules in general. It gives the option to write own ideas for improvement in a free-text field in case some characteristics were not mentioned. The questionnaire can be found in the appendix.

The survey had 166 answers from all year groups as it can be seen in Figure 1. From the answers, five characteristics could be extracted that are used in the evaluation function. For a characteristic, there can exist different preferences. An overview of these characteristics and preferences is displayed in Table 1.

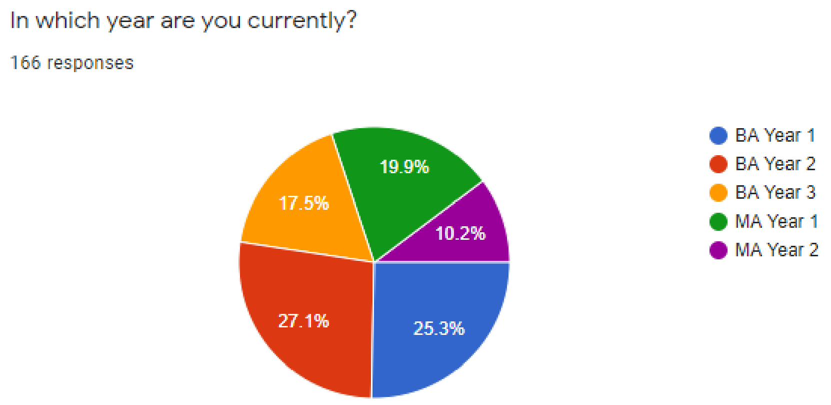


Figure 1: Division of the survey answers in year groups.

Characteristic	Preferences
Maximum hours of lectures per day	8, 6 or 4
Starting time	08:30, 11:00, 13:30
Free timeslots between lectures	0 or 1
Same schedule every week	true, don't care
Day off	Mon, Tue, Wed, Thu, Fri or None

Table 1: Preferences that were used for the evaluation function.

Two of the questions from the survey were not used for the evaluation: Firstly the question if students prefer to stay in the same room because most students were neutral about it and the results were very symmetric. Secondly the question if practicals and lectures should be scheduled on the same day. The results show that the opinions on this question vary a lot which would lead to a similar score no matter if they are on the same day or not. Moreover, in the used data structure the schedules are modeled in a way that currently does not distinguish between practicals and lectures. The survey results on these questions can be found in Figure 2.

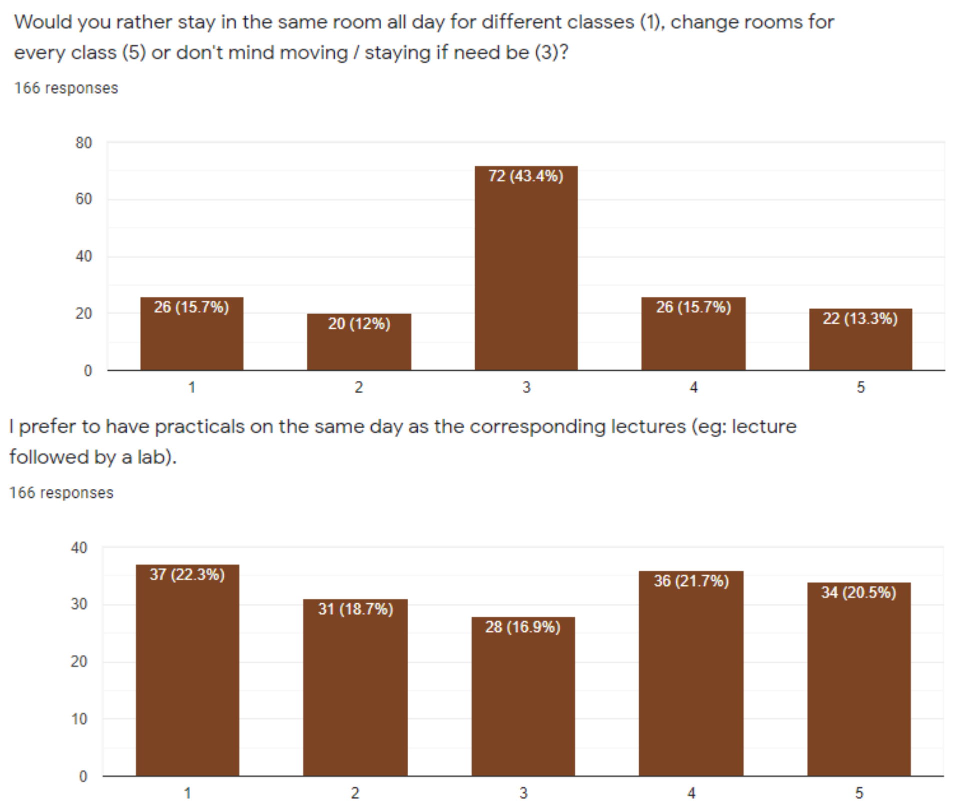


Figure 2: Answers to the questions on rooms and practicals that were not used in the evaluation function.

5.2 Evaluation Score Implementation

The evaluation was implemented in the form of a function that returns a score depending on the extend to which the soft constraints are satisfied. Checking the hard constraints is optional and can be turned on or off. This is to make the algorithm run faster if the algorithm does not produce infeasible solutions by design.

Schedules created are analyzed per year group to calculate the score, except for the regularity constraint, where each course can be regarded separately. That view per year group does not exactly reflect the schedules of each individual student, because there can be a different combination of electives or deviations from the study plan. This is to simplify the computation and also due to lack of information given.

For each preference, the survey gives the fraction of students that have that preference ($w_p(p), \forall\, \text{preferences}\, p$). These fractions are normalized to make the sum of preference fractions for a characteristic sum up to 1. This was not always the case for example when multiple answers could be selected. Moreover, the survey asked about the importance of the different characteristics. These weights ($w_c(c), \forall \text{characteristics}\, c$) are also taken into account. All fractions are stored separately from the code in a json-File such that it is possible to change them easily.

In the code, the fraction that satisfies a preference ($sat(p), \forall\, \text{preferences}\, p$) is calculated for each preference and combined in the following way:

$$evaluation\, score = \sum_{c \in \text{characteristics}} w_c(c) \cdot \sum_{p \in \text{preferences of c}} sat(p) \cdot w_p(p)$$

Technically, this allows an evaluation score of a schedule in $[0, 1]$ but as the preferences are often contradicting, these extreme values are not possible. To have benchmark schedules, it is also possible to evaluate any schedule that was not produced by any of the algorithms as long as it is available in the defined input format. This helps to get a benchmark of the evaluation score.

6 Approach / Algorithms

In this section, an overview is given of the framework that is used. Subsequently, the baseline feasible algorithms are explained, after which the used metaheuristic algorithms and ILP are explained in greater detail. Lastly, the software usability is explained for potential users.

6.1 Framework

With variations in courses, staff members, students and locations each period, scheduling software needs to rely on a robust framework in order to function properly. The data put into this framework must also be fairly easy to access and modify by the end-user. For data input, an end-user can provide information regarding the period in a Microsoft Excel workbook. Within the workbook, data is split up into five sheets pertaining to course information, room availability and capacity, information regarding start and end dates of the period and dates reserved for holidays.

Processing the input data is the next step, carried out by the *ConstraintParser*, as seen in an illustration of the framework in Figure 3 below. Before the data can be used by the algorithms, the data is stored in a dataframe, as String values or numerical values. An exception to this is the data regarding time and date. These are converted to the timestamp format. Days from the input not marked as a holiday or weekend are split up into 4 timeslots, which corresponding to the 2-hour intervals in which single classes are scheduled. This standard data format is used by all scheduling algorithms described in the following subsections. The workings of the output and GUI are described in Section 6.5.

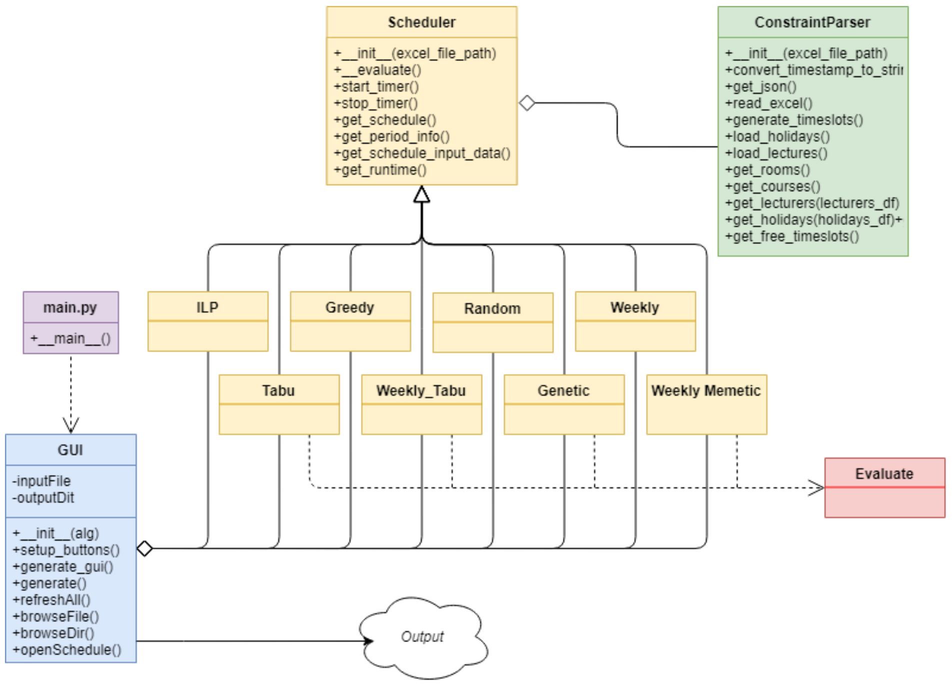


Figure 3: Framework UML Class Diagram

6.2 Baseline Greedy Algorithms

In order to measure the performance of the algorithms and heuristics investigated, a baseline has to be set as a lower bound for the algorithms. If the algorithms investigated attain scores which are worse than the baseline scores, they can be dismissed. The random and greedy algorithms created only take hard constraints into account. Both of these algorithms place classes in whichever order these are provided, unless one of the constraints is violated, in which case the next class of a specific course which does satisfy all hard constraints is scheduled. If at a particular timeslot no class can be scheduled which satisfies all hard constraints, this timeslot is left blank.

The greedy algorithm tries to place all classes of a single course before moving on to the next course. As an alternative, the random algorithm randomizes this order of classes and simply schedules the next available class, regardless of the course it belongs to. While the random algorithm picks a random available classroom for each class, the greedy algorithm schedules classes in the first available classroom.

A third baseline algorithm is the "weekly" greedy algorithm, which is a modified version of the regular greedy algorithm described above. The weekly algorithm tries to schedule a single week of courses, and then attempts to repeat this schedule for each of the remaining weeks in the period. The regular greedy and random algorithms both try to schedule all classes for the entire period at once. In the weekly algorithm, the number of contact hours per course is divided by the number of weeks; these classes are then scheduled in a single week. Then, the same technique is applied for scheduling courses as the regular greedy algorithm. If there is a scheduling conflict at a later date (for example, due to lecturer unavailability), classes are skipped just like in the regular greedy algorithm.

6.3 Metaheuristics

In this project, different metaheuristics are implemented. A metaheuristic is a certain strategy to guide a search process, given a solution, or set of solutions. These types of algorithms use techniques that have proved to be useful in finding good quality results in combinatorial optimization problems. It is not problem-specific and the framework can be applied in many situations. The goal in such an algorithm is to efficiently explore the search space to find a near-optimal solution. Many types exist in the literature, and three of them are explored in this project: tabu search, genetic algorithm, and memetic algorithm. The first is a single-solution based technique, and the other two are population-based techniques.

6.3.1 Tabu Search

Tabu search works based on a local search that starts with a solution, looks at the neighborhood of the solution and moves to the best neighbor. Thereby it can move closer towards an optimum solution with every iteration. The special feature about Tabu search is that it has a so called "tabu list" of solutions that were already considered. These solutions are ignored when looking at the neighborhood to avoid repeating the same steps. As it goes to a new solution in every step it can also escape local optima. [13]

[14] shows how the search algorithm can be applied to a timetable scheduling problem. An initial schedule is needed as input to the algorithm. For this purpose, one of the

simple baseline algorithms described in Section 6.2 is used as a default to be in principle able to test it as a standalone algorithm to find a schedule. It is however also conceivable to start with an already near-optimum schedule and use the algorithm for fine tuning.

The neighborhood of a schedule consists of schedules that are obtained by a “simple move” where a single lecture is moved to another timeslot. As the exploration of the whole neighborhood is computationally too expensive, a small number of lectures is picked randomly and each possible “simple move” that produces a feasible schedule is considered. The only thing that might be changed about the lecture is the room to allow more schedules in the neighborhood. [14]

The best solutions are determined using the evaluation score of the soft constraints. This is done for a fixed number of iterations but it is also possible to define a threshold for the evaluation score. The implemented algorithm only uses a very basic tabu list as memory where a fixed number of considered schedules is stored and the oldest is removed when the list exceeds its maximum size. As it only does few changes to the schedule in each iteration it is expected that the transformation is rather slow. For the same reason it is difficult for the algorithm to make big changes without temporarily decreasing the evaluation score. Therefore, it benefits from a good initial schedule.

Weekly Tabu Search To mitigate the problem described above without having to use the other algorithms, it is combined with the baseline Weekly algorithm explained in Section 6.2. This is implemented by first greedily generating a schedule for one week and optimizing it using the Tabu search. As the schedule is smaller, the search goes much faster. Then, the optimized schedule is copied to every week. Infeasible lectures are removed and greedily put to possible timeslots. This schedule is again optimized using Tabu search.

6.3.2 Genetic Algorithm

A genetic algorithm (GA) is a technique that helps to solve the problem by simulating the evolution of humans or the creatures in general (based on Darwin’s theory of natural evolution) under predefined conditions of the environment. A genetic algorithm is a sub-branch of evolutionary algorithms (EAs). The process of natural selection derives from original potential solutions, GA conducts a search on the solution space by creating new generations of solutions that are better (more optimized) than the old ones. The process of generating a new solution is conducted through the selection, crossover, and mutation from the original solution. The new solution population then undergoes the evolution process: within each generation regenerate solutions that are feasible, while the bad solutions are eliminated.

The basic idea of applying genetic algorithm for generating a timetable is followed by the below steps:

1. Initialize a random population in which each chromosome represent for a possible solution (timetable).
2. Estimate the fitness value of each chromosome in the current population based on the constraints.
3. Select a pair of parents through its fitness score using the tournament selection method.
4. Apply crossover and mutation with the probability of 80% and 1% respectively, thereby creating new generations of chromosomes from selected parents. These offspring will be added to the current population to replace the "bad" individuals.
5. Repeatedly these processes until finding the feasible solution or reaching a certain number of generations.

To be more specific, first of all, a random initial population is generated. Each individual in the population is a potential solution (timetable) containing genes, each gene includes an information course, timeslot, and room. Then, calculating the fitness score of each individual base on the constraints (both hard and soft constraints described in Section 5) to select a pair of parents for reproduction next generation. There are many different strategies for selection such as roulette wheel selection, ranking selection, and tournament selection. In this project, the tournament selection method is chosen to deal with this task. After choosing two chromosomes parents, two new offsprings then are created by crossing of these parents' genes (an information of course, timeslot and room) with the crossover probability of 80%. In the next step, a mutation can happen in the new generated descendants with 1% probability. Finally, comparing the fitness score of these new individuals with "bad" individuals in current population if the new generation individuals are better, adding them to the next generation and removing these "bad" ones. The iteration stops only if the feasible timetable found or reaching a defined number of maximum generations.

At the end of the process, depending on how much contact time needed for these courses, the resulting schedule then is applied for every week. However, since the genetic algorithm is a random search algorithm, the final result is not always guaranteed optimal solution.

6.3.3 Memetic Algorithm

The other metaheuristic that is considered is a memetic algorithm. This algorithm is a combination of a genetic and hill climbing or local search algorithm. In this approach, schedules are created for one week, which are copied to form a schedule for a period. The structure of the algorithm is similar to that of a genetic algorithm and the pseudocode is given in Algorithm 1.

Algorithm 1 Memetic algorithm

Input: Problem instance $\mathcal{I}$
Parameters: Population size n, number of iterations m, number of local search iterations ℓ, mutation probability p_m, tournament size τ
Returns: Best solution found s_{best}

```
1: function MEMETIC($\mathcal{I}$)
2:     init:
3:         initialize population $\mathcal{P}$ of size $n$
4:         perform local search on each individual in $\mathcal{P}$
5:     while termination condition not met do:
6:         select two parents using tournament selection with size $\tau$
7:         create child using crossover
8:         apply mutation on child with probability $p_m$
9:         apply local search on child
10:        replace individual in $\mathcal{P}$ with the lowest score by the child
11:    return:
12:        individual with the best score in $\mathcal{P}$
13: end function
```

The steps are explained further as follows:

1. To initialize a population with n solutions, one solution is generated using a greedy approach. The remaining $n-1$ solutions are generated in a random way.
2. The local search procedure uses simple destroy and repair operators based on the worst removal and basic greedy heuristic in [15].
3. In the selection step, parent solutions are selected by using tournament selection. First, c candidates are randomly sampled and the candidate with the highest score is chosen to be a parent. This is repeated until two parents are chosen.

4. In the regeneration step, a child solution is created using a crossover operator based on the one in [16]. However, instead of selecting a parent to allocate the corresponding time slot for events individually, a parent is chosen for all events per year group. This is to avoid having a year group to have two events in one time slot.
5. For the mutation procedure, a year group is chosen at random to have all the corresponding events removed from the solution. Then, they are re-inserted by using greedy insertion from local search.

6.4 Integer Linear Programming

This section describes an approach to use an Integer Linear Program (ILP) to solve the scheduling problem. A linear program (LP) seeks to optimize an objective function, typically representing profit or score, subject to linear constraints. These constraints set bounds on the values of the variables in the objective function. An Integer Linear Program (ILP) is a form of linear program where some of these variables are restricted to integer or binary (decision) values. When creating schedules with an ILP, heavy inspiration from the course Planning and Scheduling was drawn. It turns out that it suffices to restrict the ILP to exclusively use integer variables and binary variables.

First, the notation is given for sets, parameters and variables. Second, a mathematical formulation is given of the ILP with explanations of the constraints in greater detail. Lastly, a brief overview is provided of the complexity and performance of the ILP.

6.4.1 Notation

To set up a mathematical formulation, some sets and parameters are defined. Then, the variables can be defined to set up the objective function and the constraints.

Sets Given a problem instance of a certain period, sets are defined as follows:

- $\mathcal{D}$: set of all days d;
- $\mathcal{D}_w$: {Monday, Tuesday, Wednesday, Thursday, Friday}
- $\mathcal{S}$: set of starting times, $\mathcal{S} = \{8{:}30, 11{:}00, 13{:}30, 16{:}00\}$;
- $\mathcal{Y}$: set of all year groups y;
- $\mathcal{R}_{i,d,y}$: repeating timeslots which start at $i \in \mathcal{S}$ and are on day $d \in \mathcal{D}_w$ for year group $y \in \mathcal{Y}$;

- $\mathcal{T}$: set of all timeslots t;
- $\mathcal{T}_i$: subset of $\mathcal{T}$, of timeslots starting at $i \in \mathcal{S}$
- $\mathcal{L}$: set of all lecturers l;
- $\mathcal{C}$: set of all course IDs c;
- $\mathcal{C}_y$: set of course IDs corresponding to year group $y \in \mathcal{Y}$
- $\mathcal{C}_l$: set of course IDs of courses taught by lecturer $l \in \mathcal{L}$

Parameters The following parameters are derived from the problem instance. The weight parameters are established using the results of the survey.

- h_c: number of contact hours of course $c \in \mathcal{C}$;
- $u_{t,c} = \begin{cases} 1 \text{ if course } c \text{ cannot be taught at timeslot } t, \\ 0 \text{ otherwise.} \end{cases}$
- σ_i: Weight for preferred lecture start $i \in \mathcal{S}$;
- γ_j: Weight for the maximum of j preferred lectures per day, $j \in \{1, 2, 3\}$
- ρ_d: Weight for the preferred weekday $d \in \mathcal{D}_w$ without classes.

Auxiliary variables

- $r_{i,d,y}$: Penalty for classes scheduled on repeating timeslot $i \in \mathcal{S}$ on day $d \in \mathcal{D}_w$ for year group $y \in \mathcal{Y}$, enforcing regularity;
- s_i: Number of lectures scheduled at time $i \in \mathcal{S}$;
- $c_{j,d,y} = \begin{cases} 1 \text{ if more than } j \text{ classes are scheduled on day } d \in \mathcal{D} \text{ for year group } y \in \mathcal{Y} \\ 0 \text{ otherwise;} \end{cases}$
- $o_{d,y} = \begin{cases} 1 \text{ if day } d \in \mathcal{D} \text{ has no events scheduled for year group } y \in \mathcal{Y} \\ 0 \text{ otherwise;} \end{cases}$
- $l_{d,y}$: the number of scheduled lectures for year group $y \in \mathcal{Y}$ on day $d \in \mathcal{D}$.

Decision variable

$$x_{t,c} = \begin{cases} 1 \text{ if course } c \text{ is scheduled at timeslot } t, \\ 0 \text{ otherwise.} \end{cases}$$

The variables are differentiated between decision variables and auxiliary variables. The decision variables have no direct effect on the objective function themselves. When translating a solution from the ILP to a schedule, courses are scheduled at timeslots where the respective decision variable has value 1. The auxiliary variables can either be integer or strictly binary variables and represent a property of the schedule, for example if a class is scheduled at 8:30. These variables are added (or subtracted) with respective weights in the objective function and thus determine the quality of the schedule.

6.4.2 Mathematical formulation of the ILP

The following shows a mathematical representation of the ILP. The constraints of the ILP are divided into two classes, representing the hard (Constraint (2)-(5)) and soft constraints (Constraint (6)-(9)). Hard constraints solely ensure that the generated solutions are feasible schedules (for example, that the amount of contact hours are met for each course), and thus do not require auxiliary variables. Soft constraints determine the quality of the schedule. These constraints implement the quality measures determined from the survey described in Section 5 and assign values to the auxiliary variables based on the values of the decision variables. The soft constraints include: The repetitiveness of the schedule, the time lectures are scheduled, days which are off, and the number of lectures on a day.

The formulation is followed by a detailed description of the implementation and functionality of each constraint.

$$\min \sum_{i\in\mathcal{S}} \sum_{d\in\mathcal{D}_w} \sum_{y\in\mathcal{Y}} r_{i,d,y} - \sum_{i\in\mathcal{S}} \sigma_i s_i - \sum_{j\in\{1,2,3\}} \sum_{d\in\mathcal{D}} \sum_{y\in\mathcal{Y}} \gamma_j c_{j,d,y} - \sum_{d\in\mathcal{D}} \sum_{y\in\mathcal{Y}} \rho_d (1 - o_{d,y}) \tag{1}$$

$$\text{s.t.} \sum x_{t,c} = h_c, \quad \forall c \in \mathcal{C}, \tag{2}$$

$$\sum_{c\in\mathcal{C}_y} x_{t,c} \leq 1, \quad \forall t \in \mathcal{T}, \quad \forall y \in \mathcal{Y}, \tag{3}$$

$$\sum_{c\in\mathcal{C}_l} x_{t,c} \leq 1, \quad \forall t \in \mathcal{T}, \quad \forall l \in \mathcal{L}, \tag{4}$$

$$u_{t,c} x_{t,c} = 0, \quad \forall t \in \mathcal{T}, \quad c \in \mathcal{C}, \tag{5}$$

$$M \cdot r_{i,d,y} \geq \sum_{t\in\mathcal{R}_{i,d,y}} \sum_{c\in\mathcal{C}_y} x_{t,c}, \quad \forall i \in \mathcal{S}, \quad \forall d \in \mathcal{D}_w, \quad \forall y \in \mathcal{Y}, \tag{6}$$

$$s_i = \sum_{t\in T_i} x_{t,c}, \quad \forall i \in \mathcal{S}, \quad \forall c \in \mathcal{C}, \tag{7}$$

$$j + M \cdot c_{j,d,y} \geq l_{d,y}, \quad \forall d \in \mathcal{D}, \quad \forall y \in \mathcal{Y}, \quad \forall j \in \{1,2,3\}, \tag{8}$$

$$M \cdot o_{d,y} \geq l_{d,y}, \quad \forall d \in \mathcal{D}, \quad \forall y \in \mathcal{Y}, \tag{9}$$

$$x_{t,c} \in \{0,1\}, \quad \forall t \in \mathcal{T}, c \in \mathcal{C}, \tag{10}$$

$$c_{j,d,y} \in \{0,1\}, \quad \forall j \in \{1,2,3\}, \quad \forall d \in \mathcal{D}, \quad \forall y \in \mathcal{Y}, \tag{11}$$

$$o_{d,y} \in \{0,1\}, \quad \forall d \in \mathcal{D}, \quad \forall y \in \mathcal{Y}, \tag{12}$$

$$r_{i,d,y} \in \mathbb{N}^{+}, \quad \forall i \in \mathcal{S}, \quad \forall d \in \mathcal{D}_w, \quad \forall y \in \mathcal{Y}, \tag{13}$$

$$s_i \in \mathbb{N}^{+}, \quad \forall i \in \mathcal{S}. \tag{14}$$

(2) corresponds to the an equality constraint which ensures that every course is scheduled for exactly the amount of contact hours required. In practice, a constraint is created for every course which sums the decision variables for all timeslots. The sum of all timeslots for a course has to be equal to the number of specified contact hours for the respective course.
(3) ensures that no two courses of the same year are scheduled at the same time. For every timeslot, the decision variable of courses which are in the same year are summed. To guarantee that no two classes are scheduled at the same time, the sum of all these courses has to be at most one.
(4) restricts the number of courses a lecturer can teach at any given slot timeslot. It is quite similar to the second constraint, but instead of summing over all courses in every year, we sum over all courses that are taught by a given lecturer in every timeslot. Note that if a lecturer teaches only one course, this constraint is already satisfied by definition and thus can be discarded.
(5) ensures that if a lecturer is absent, they are not scheduled to teach any courses during their absence. This can simply be achieved by setting all decision variables for courses taught in this interval by this lecturer to 0.
(6) is a soft constraint which penalizes irregularities in the schedule. The right hand side sums over all repeating timeslots of the schedule (e.g. every Monday at 8:30) for every timeslot in the first week. For all these options, a constraint is added and the *Big-M* method is applied [17] with a binary decision variable $R_{i,d,y}$ on the left hand side. Since at most eight weeks are scheduled in any period, M is set to be equal to 8, since this increases the computational efficiency of the program. The sum of all $r_{i,d,y}$ is added

to the objective function (1) and, since this is a minimization program, functions as a penalty. Consequently, the program tries to minimize the number of r variables which are set to 1. Observe that if we schedule a single class at a timeslot, we need to set $r = 1$ to satisfy the constraint. If more than one class is scheduled at a timeslot, the constraint is still satisfied with $r = 1$. If the ILP does not schedule courses routinely, multiple $r_{i,d,y}$ have to be set to 1 to obtain a feasible solution. Therefore, the ILP tries to construct the schedules in a repetitive manner to minimize the number $r_{i,d,y}$ variables which have to be set to 1.

(7) decreases the value of the minimizing ILP depending on the timeslot where a lecture is scheduled. The s_i variables are subtracted from the objective function with respective weights obtained by the survey. Therefore, the ILP is more likely to schedule classes at timeslots preferred by the students, since this decreases the score of the objective function.

(8) counts the number of lectures scheduled on a single day. In the survey, students were asked to set an upper bound on the maximum number of lectures per day they would tolerate. This is modeled by variables $c_{i,d,y}$, which are added to the objective function. The ILP tries to set as few as possible to 1. The right hand side counts the number of lectures scheduled per day. The left hand side multiplies the binary variable with M and adds either 1, 2, or 3, corresponding to the number of lectures scheduled on a single day. Since at most four classes can be scheduled at one time, M can be chosen to be equal to 4. Assume a fixed day d and a fixed year group y. If two lectures are scheduled on the day, we have to set $c_{1,d,y}$ to 1 but not $c_{2,d,y}$ or $c_{3,d,y}$. Observe that if we schedule two events, we schedule less than 6 and also less than 8 hours on a day. Students who answered that they would prefer to have 6 or less hours per day should also be satisfied with 4 or less hours on a day. It turns out that due to the other properties of the ILP, this constraint does not have a high impact on the resulting solution.

(9) counts the number of free days available and can, with weight selection, also determine which days have no lectures scheduled. For every day d, a constraint is added. Here, the *Big-M* method is applied and on the left hand side a binary variable is multiplied with M. With the same reasoning from constraint (7), M is set to be 4, corresponding to the maximum number of concurrently scheduled courses. If a lecture is scheduled on day d for year group y, then $o_{d,y}$ must be set to 1. Since $(1 - o_{d,y})$ is subtracted from the objective function, the ILP strives to schedule as many days off as possible. Setting $o_{d,y}$ to zero can be achieved by not scheduling any classes on a day as a single class forces

$o_{d,y} = 1$ and thus it does not decrease the value of the objective function.
(10)-(14) define the range of the auxiliary and decision variables.

6.4.3 Complexity and Analysis

It is apparent that the ILP grows quickly with respect to the number of timeslots per day, the number of courses to be scheduled and the length of the period. Running the ILP on a test schedule with 21 courses, 4 timeslots per day for 8 weeks of 5 days each, results in a total of 3360 decision variables and 1672 auxiliary variables. A total number of 3697 constraints are constructed, out of which 1857 model the hard constraints. The number of constraints may vary considerably per period depending on the input data, and only serve the purpose of providing a rough estimate of the dimensions of the ILP to the reader.

In this project, an open source mixed integer program solver (Python MIP) was used to compute a solution to the ILP. In the analysis, the solver will be treated as a black box and only general properties will be analysed. When applied to this problem, the ILP solver that is used cannot compute an optimal solution. The reason for this is twofold. For one, the solution set is pareto-optimal. Generally, this does not hinder an ILP sovler in finding an optimal solution, but due to the high dimensionality of the program, there is no apparent structure in the pareto-optimal set [18]. The second, and more severe reason is that the methods employed to solve the ILP generally uses the strong duality theorem [19], which states that any feasible solution of the dual acts as an upper bound for the primal. Due to the *Big-M* method which is used to design the soft constraints, we can observe that the feasible solution found in the evaluation process is non-integer. This is also the case if no weights are applied to the auxiliary variables in the objective function. In this case, only integer solutions are valid optimal solutions, and due to the size the algorithm it cannot narrow down the dual solution to an integer value. Therefore, the solver gets stuck. A workaround for this is to limit the time that the solver has and prioritize to find a feasible solution. Despite the size of the ILP, in 100 bazillion runs, a feasible solution was found after 2.23 seconds on average. Since the resulting feasible solution does not obtain a high score in the evaluation, the solver is given a total of 30 seconds to find a feasible solution and to improve it before it returns a result.

In all, the ILP is able to produce feasible schedules of relatively high quality, and is thus a powerful tool to solve the scheduling problem. Additionally, it is highly flexible by steering the parameters of the auxiliary variables. Increasing the weights of the variables respective to constraint (9) for example, results in a schedule which tries to minimize the number of lecture days. The desired starting time for lectures can be steered by changing the weights of the variables in constraint (7). This is because the variables then have a greater impact on the objective function.

6.5 Usability

Once the previous algorithms were working, the next step was to improve the software usability since at the beginning the debugging operations consisted in raw JSON output, which is clearly not optimal to deliver to the DKE. To do this a GUI (Graphical User Interface) was created, in which the user can perform five actions: (1) select an excel file as input, (2) specify the output directory location to where the final schedule will be delivered, (3) select the desired algorithm to work with, (4) generate the schedule with the previous parameters, and (5) open a saved timetable.

The GUI was designed using the popular *Qt* toolkit that provides a powerful cross-platform application (see Figure 4) so that there is no need to worry about the destination operating system. Even though the set of actions you can perform with this framework is relatively low, a GUI like the one presented here provides an inexperienced user sufficient tools to get it to work. In order to do this correctly, the user interface usability guidelines were followed in the design, such as: abstracting the GUI from the code, use proximity rules, use a familiar display order (top to bottom), etc.

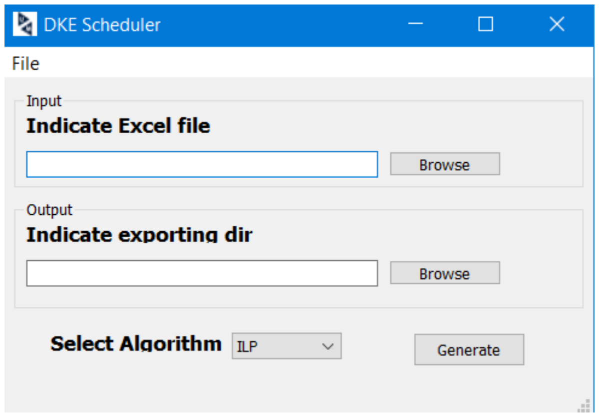
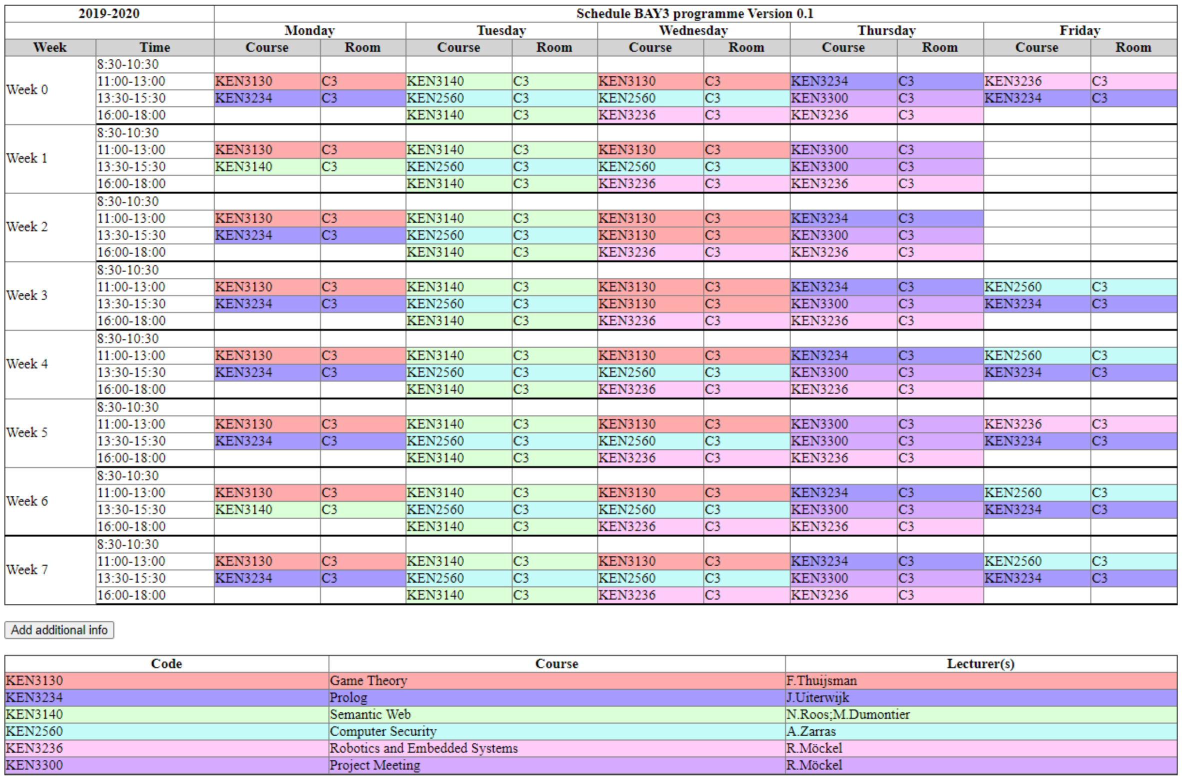


Figure 4: Final version of the GUI.

When a user clicks on “Generate”, the framework will call the selected algorithm and run it. Once it finishes, the resulting JSON file with the final schedule will be parsed into a table in HTML format. The mentioned HTML will be showed back to the user using the default computer browser as can be seen in Figure 5. It contains the final generated timetable divided by year groups and weeks. The format and coloring of such timetables is based on real previous ones presented to the students by the DKE. This way it is familiar and easy to adapt to a final product that can be used in a real scenario.



2019-2020		Schedule BAY3 programme Version 0.1									
		Monday		Tuesday		Wednesday		Thursday		Friday	
Week	Time	Course	Room	Course	Room	Course	Room	Course	Room	Course	Room
Week 0	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3	KEN3236	C3
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 1	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3300	C3		
	13:30-15:30	KEN3140	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3		
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 2	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3		
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN3130	C3	KEN3300	C3		
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 3	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3	KEN2560	C3
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN3130	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 4	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3	KEN2560	C3
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 5	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3300	C3	KEN3236	C3
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 6	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3	KEN2560	C3
	13:30-15:30	KEN3140	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		
Week 7	8:30-10:30										
	11:00-13:00	KEN3130	C3	KEN3140	C3	KEN3130	C3	KEN3234	C3	KEN2560	C3
	13:30-15:30	KEN3234	C3	KEN2560	C3	KEN2560	C3	KEN3300	C3	KEN3234	C3
	16:00-18:00			KEN3140	C3	KEN3236	C3	KEN3236	C3		

Add additional info

Code	Course	Lecturer(s)
KEN3130	Game Theory	F.Thuijsman
KEN3234	Prolog	J.Uiterwijk
KEN3140	Semantic Web	N.Roos;M.Dumontier
KEN2560	Computer Security	A.Zarras
KEN3236	Robotics and Embedded Systems	R.Möckel
KEN3300	Project Meeting	R.Möckel

Figure 5: Resulting generated timetable for BAY3 year group using test data from past year and the ILP algorithm.

The resulting timetables can now also be modified within the HTML with the help of a JavaScript frontend. The user can move courses to other time slots if needed and to facilitate this task, the ones that it cannot be moved into due to collisions between courses or professors, are highlighted in red. At this point it is also possible to download these timetables. The output is saved into a JSON file for convenience, so that it can be loaded later as stated at the beginning of this section. Since the working environment would be the user's default browser, it has a built-in function to print the HTML site with the timetables, so that the framework does not need to handle it. A complete user manual, explaining how to work with the product can be found in Appendix A.

7 Results

While the user interface is designed in a way that allows to try all algorithms and manually choose which one to work with, it is still interesting to see how the algorithms compare in terms of runtime and solution quality. As the framework and the algorithms are strongly tailored to DKE schedules, input for experiments must be as close as possible to them.

In detail, the test data consists of a sample input that is approximately based on last year's schedule for Period 1 and was used during development. Another input file was generated based on the data of last year's schedule for Period 2. Some of the constraints are not known, mainly the lecturer unavailability that can differ. For this input, the manually created schedule was also converted to the JSON-format to compare its evaluation score to those of the generated schedules. Compiling an input for a complete schedule like this is time consuming, which is why it was not done for more periods. Another point to is that not every element of a schedule can be modeled in the data structure and thus be considered by the algorithms. This includes for example scheduling courses for two groups or with two lecturers where only one of them has to be present at a lecture.

Table 2 shows the average runtime and evaluation score for each algorithm on the different input data. Each algorithm was run 10 times on a normal average computer and the mean of runtimes and evaluation scores was computed to make it less dependent on CPU load and chance for the algorithms that include randomization. The results give an idea of how the algorithms perform, but they have to be viewed with caution. For the runtime, it is important to say that this was never a main priority as long as the algorithms finish in reasonable time. Therefore, the code is not optimized on that. Also, the number of iterations and amount of output to the console is not consistent. The evaluation is based on the preferences of the students, but this was not validated again in the end. It is conceivable that the evaluation score does not take something into account that is unconsciously very important for the quality.

Input	**Algorithm**	**Runtime** *(in seconds)*	**Evaluation Score**
Sample Period 1	Greedy	0.10	0.2579
	Random	0.06	0.2498
	Weekly	**0.05**	0.4362
	Genetic	95.01	0.5659
	Memetic	390.36	0.6117
	Tabu	79.39	0.4305
	Weekly Tabu	130.64	**0.7060**
	ILP	39.90	0.6578
Period 2 (*without lecturer unavailabilities*)	Greedy	0.07	0.2271
	Random	0.07	0.2332
	Weekly	**0.04**	0.3882
	Genetic	347.64	0.5074
	Memetic	:(*	:(*
	Tabu	45.35	0.3526
	Weekly Tabu	54.09	**0.5469**
	ILP	40.78	0.5101
Period 2 (*with lecturer unavailabilities*)	Greedy	0.13	0.2271
	Random	0.13	0.2366
	Weekly	**0.075**	0.3970
	Genetic	262.67	0.5144
	Memetic	:(*	:(*
	Tabu	57.33	0.3565
	Weekly Tabu	72.01	**0.5613**
	ILP	42.29	0.5100
Period 2	Manually	Long	0.4628

Table 2: Results of the algorithms regarding runtime and evaluation score. Best results for each sample input are highlighted in bold. *Note: the results for Period 2 are missing for the memetic algorithm due to an error that could not be resolved within the given timespan.

8 Conclusion

In this project, we proposed a relatively ambitious goal: generating feasible and qualitative schedules for DKE, trying to maximize student satisfaction. As a start, we demonstrated the potential of this research by constructing a framework from scratch with the most popular existing approaches for scheduling tasks. By conducting a survey amongst DKE students, we obtained useful data regarding their preferences used to improve the schedules. In order to weigh all constraints attached to our scheduling problem, we presented an evaluation function (Section 5.2) that showed very promising and reliable results with a majority satisfaction technique.

During this project we implemented scheduling algorithms from various fields, as we attempted to solve the problem from different angles. Some are straightforward and served as a benchmark and others employed techniques used in the field of artificial intelligence and operations research. We implemented and tested a basic greedy approach, metaheuristic approaches that include tabu search, a genetic and a memetic algorithm; and an ILP approach. This variety helped us to understand the pros and cons of every approach when facing different real-life scenarios (Section 7).

The silver lining of our framework is that many algorithms can live together within the same input and output format. This allows an opportunity for future work to improve upon our results with more algorithms without the need of starting again from scratch. Additionally, the application created during this project reduces the complexity of schedule creation for the department scheduler.

9 Future work

Creating optimal university timetables is an ongoing field of research. Since each academic period schedule is unique with variations in the variables and constraints, the quest for an algorithm which can guarantee optimality, even approximately, is endless.

While the program created for this specific use case accurately models the constraints and variables involved to create optimal and feasible schedules, this has not been verified in practice. The current model assumes the constraints and variables provided by the scheduler at DKE. Despite this, it is possible that there are variables and constraints which have not yet been fully modeled yet. Further research needs to be carried out to establish whether the proposed schedules are indeed optimal when applied in practice. This would also require further testing among the students subject to the schedules. In the future, it may also be worthwhile to incorporate constraints reflecting the preferences of the staff members. While the inclusion of these additional constraints may yield a lower overall score compared to the current model, it is anticipated that more people will comparatively be satisfied with the schedule.

A limitation of the model is that it does not “hard-code” intervals reserved for projects in the bachelor program. This limitation can easily be overcome by adding a placeholder “dummy” course, assigned to “dummy” staff members who are only available in these intervals. Support for implementing these projects directly would be preferred over this workaround.

Besides that, reducing running time and improving an adjustment function are also essential focused tasks in the future with the aim that users have more flexibility to customize the timetable and spend less time creating a schedule. Furthermore, due to the outbreak of COVID-19, several requirements for the timetable of the upcoming semester should be considered such as an extra timeslot from 18:00 to 20:00.

Moreover, the application created in the process of this research project can be extended in a few ways. The program can be made into an executable for easier use. It can for instance incorporate a function allowing staff members to input data regarding their availability in a web format directly, so that the scheduler does not need to input this data manually in an Excel file.

Lastly, output website should be more robust. Editing, saving or swapping courses has not been tested extensively. To prevent failure, assessment test can be written and the error logging could be improved.

References

[1] George M White and Junhan Zhang. Generating complete university timetables by combining tabu search with constraint logic. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 187–198. Springer, 1997.

[2] Tim B Cooper and Jeffrey H Kingston. The complexity of timetable construction problems. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 281–295. Springer, 1995.

[3] Hamed Babaei, Jaber Karimpour, and Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86:43–59, 2015.

[4] Pariwat Khonggamnerd and Supachate Innet. On improvement of effectiveness in automatic university timetabling arrangement with applied genetic algorithm. In *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pages 1266–1270. IEEE, 2009.

[5] Halvard Arntzen and Arne Lokketangen. A local search heuristic for a university timetabling problem. *nine*, 1(T2):T45, 2003.

[6] Andrea Schaerf. Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 29(4):368–377, 1999.

[7] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 104–117. Springer, 2000.

[8] Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, 30(1):167–190, 2008.

[9] Sophia Daskalaki, Theodore Birbas, and Efthymios Housos. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*, 153(1):117–135, 2004.

[10] Sophia Daskalaki and Theodore Birbas. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, 160(1):106–120, 2005.

[11] Mike Wright. School timetabling using heuristic search. *Journal of The Operational Research Society - J OPER RES SOC*, 47:347–357, 03 1996.

[12] Fred Shen Maxime Laschet Anton Bulat, Fatimah Mulan Ahmed. Dke scheduling project. *https://project.dke.maastrichtuniversity.nl/studentprojects/?p=244*, 2015.

[13] Hamed Babaei, Jaber Karimpour, and Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86:43–59, 2015.

[14] Çağdaş Hakan Aladağ and Gülsüm Hocaŏ. A tabu search algorithm to solve a course timetabling problem. *Hacettepe Journal of Mathematics and Statistics Volume*, 36:53–64, 01 2007.

[15] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[16] Shengxiang Yang and Sadaf Naseem Jat. Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):93–106, 2010.

[17] PRASHANT P Bedekar, SUDHIR R Bhide, and VIJAY S Kale. Optimum time coordination of overcurrent relays in distribution system using big-m (penalty) method. *WSEAS Transactions on Power Systems*, 4(11):341–350, 2009.

[18] Noureddine El Karoui et al. High-dimensionality effects in the markowitz problem and other quadratic programs with linear constraints: Risk underestimation. *The Annals of Statistics*, 38(6):3487–3566, 2010.

[19] Jaroslav Ramík. Duality in fuzzy linear programming: some new concepts and results. *Fuzzy Optimization and Decision Making*, 4(1):25–39, 2005.

[20] Hadrien Cambazard, Emmanuel Hebrard, Barry O'Sullivan, and Alexandre Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research*, 194(1):111–135, 2012.

[21] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.