Stefan Lendl

The Timetabling Problem at the CAMPUS 02 University of Applied Sciences

MASTER THESIS

written to obtain the academic degree of a

Diplom-Ingenieur

Master program *Technical Mathematics: Operations Research and Statistics*

submitted at the
Graz University of Technology

Supervisor:
Univ.-Doz. Dipl.-Ing. Dr.techn. Johannes Hatzl

Institute of Optimization and Discrete Mathematics
Graz University of Technology

Graz, August 2015

Abstract

The thesis deals with the problem of finding a timetable at the 'Information Technologies & Business Informatics' department of the CAMPUS 02 University of Applied Sciences in Graz. After a short review of the literature about timetabling problems the actual problem is formalized and is compared with similar problems analyzed in other publications. Moreover it is discussed how a suitable objective function can be found which takes into account the wishes of different stakeholders.

In the theoretical part of the thesis it is proved that the problem is NP-complete and connections to well-known graph coloring problems are emphasized. A linear integer programming formulation of the problem enables the application of general purpose solvers to find exact solutions for the problem. However, experimental results show that we cannot even hope to find a feasible timetable within acceptable running time. This shows that the problem is also computationally hard in practice.

As a consequence, the thesis presents several efficient heuristics which also work in practical applications. On the one hand construction and perturbation heuristics are suggested. On the other hand, based on these heuristics a hybrid selection hyper-heuristic using a genetic algorithm is presented.

All these heuristics are implemented and computational experiments show that it is possible to find competitive results. Especially for the hyper-heuristics including randomness seems to lead to good results.

Finally, the hyper-heuristic is generalized to a multi-objective formulation of the problem and it was again possible to obtain excellent results for this setting.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der automatisierten Stundenplanerstellung für den Studiengang 'Informationstechnologien & Wirtschaftsinformatik' an der FH CAMPUS 02 in Graz. Nach einer kurzen Literaturübersicht zum Thema Stundenplanerstellung wird das konkrete Problem formalisiert und mit bekannten Problemstellungen verglichen. Außerdem erläutern wir die Problematik, eine geeignete Zielfunktion zu definieren, die unterschiedliche Wünsche verschiedener Stakeholder berücksichtigt.

Im theoretischen Teil der Arbeit wird gezeigt, dass das vorliegende Stundenplanproblem NP-vollständig ist. Außerdem werden Zusammenhänge zu klassischen Graphenfärbungsproblemen aufgezeigt, die als Teilprobleme bei der algorithmischen Stundenplanerstellung auftreten. Eine Modellierung als ganzzahliges lineares Optimierungsproblem erlaubt den Einsatz von vorhandener Software zur Bestimmung von exakten Lösungen. Die durchgeführten Experimente zeigen jedoch, dass damit in für praktische Anwendungen akzeptabler Laufzeit keine zulässigen Lösungen erzielt werden können.

Aus diesem Grund werden in der Arbeit Heuristiken zur effizienten Berechnung von Lösungen entwickelt, die es auch in der Praxis erlauben, gute Stundenpläne zu erzielen. Einerseits werden konstruktive und perturbierende Heuristiken vorgestellt. Andererseits wird auf diese Heuristiken aufbauend eine hybride Hyper-Heuristik entwickelt, welche auf einem genetischen Algorithmus basiert.

Die vorgestellten Heuristiken wurden im Zuge der vorliegenden Arbeit implementiert und die durchgeführten Experimente zeigen, dass dadurch kompetitive Ergebnisse erzielt werden können. Weiters stellt sich heraus, dass eine geeignete Randomisierung bei der Hyper-Heuristik einen wichtigen Einfluss auf die Qualität der Lösung hat.

Schließlich wird die entwickelte Hyper-Heuristik auch auf eine multikriterielle Formulierung des Problems verallgemeinert und zeigt auch dafür hervorragende Ergebnisse für praktische Testinstanzen.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Johannes Hatzl for suggesting this topic for my master thesis and for his great support during writing this thesis.

I would also like to thank Stefan Grünwald from the CAMPUS 02 University of Applied Sciences for providing the opportunity to write my master thesis about the timetabling problem at his department and for the discussions about the problem.

Special thanks go to Silvia Poschauko for providing me with her view and expertise on the CAMPUS 02 timetabling problem.

I would also like to thank Manfred Scheucher and Paul Tabatabai for reading some parts of this thesis.

Last but not least, I want to thank my family for their support during my education and throughout my whole life.

Contents

1. Introduction

1.1. Informal Problem Description

In this thesis, the timetabling problem at the 'Information Technologies & Business Informatics' department of the CAMPUS 02 University of Applied Sciences in Graz is formalized and optimization methods are used to calculate solutions. To achieve this, the problem is modeled both as a general timetabling problem and as an integer program. For these models algorithms are designed and implemented for evaluation.

Because the program for which the timetable is generated is a part-time studies program a fixed set of weekends consisting of Friday evenings and Saturday mornings are given for planning. In addition a week consisting of Monday to Thursday, called intensive week, is given for each class to schedule the courses of the curriculum. On each of the given days 9 fixed time periods, called hours, are available for scheduling. The main constraints for planning are that at any given hour a class can only have one course scheduled, and every lecturer can only teach one course at a time.

The program consists of 5 classes, where three of them correspond to a bachelor degree program and two to a master degree program. The timetabling problem is only concerned with the first 4 classes, because the second master's class is planned independently. Divided among these classes are about 40 courses (the exact number differs each term by small changes in the curriculum) that have to be scheduled. For each of these courses there is a given number of hours that need to be scheduled. Some of the courses also contain an exam that needs a certain number of hours scheduled.

The task is to find a timetable for the classes and lecturers of the curriculum. In the CAMPUS 02 timetabling problem all students of a class take the same courses and the lecturers are already assigned to the courses. One lecturer can be assigned to teach several courses. It is possible that several lecturers are assigned to one course, meaning in practice that there are multiple groups of the course taught simultaneously. This assignment of lecturers to courses is not part of the timetabling problem.

One major aspect of this timetabling problem is that for most courses it is not feasible to plan them at single hours. It is necessary to assign several consecutive hours, called blocks, to the same course.

A timetable has to fulfill several further constraints by employment law and wishes of students and lecturers, given by the following list:

- For every day, there is only one consecutive block of hours assigned to each course.

- The lengths (number of consecutive hours) of such blocks have lower and upper bounds for each course.
- Courses have to be scheduled during a fixed time period, given by start and end dates for each course.
- For classes there is no idle time during a day.
- On each day of the intensive week, courses have to be scheduled.
- Lecturers are not allowed to teach both the last hour on Friday and the first hour on the following Saturday.
- Lecturers can define several days, called NoGo-days, on which they are not available for teaching.

Additionally, some courses contain exams in their curriculum. These exams are single blocks of predefined length. For exams there are several further constraints:

- Exams should be scheduled during the beginning of a day.
- There is only one exam allowed during every weekend.
- No exams should be scheduled during the intensive week.
- There has to be one week between the last regular course and its exam.

Every timetable has to fulfill all these constraints and schedule all courses contained in the curriculum completely. The problem is to find such feasible timetables and among all of them one that is "good". To achieve this, properties of good timetables are defined. The following is a list of such properties from the staff of the CAMPUS 02 University of Applied Sciences responsible for creating timetables:

- The number of free weekends should be maximized.
- In the timetables of each class, days should be either empty or fully scheduled.
- Hard courses should be scheduled in the early time slots.
- No idle times for lecturers.

Note that it is hard or even impossible to formally define what is meant by a good timetable. In Section 2.2 we look at different approaches to allow optimization methods to find "good" timetables using different objectives.

As part of the thesis we also provide software to solve this timetabling problem.

1.2. Overview of the Thesis

In Section 1.3 a very general version of the timetabling problem is introduced and several classical educational timetabling problems are shown to be special cases. The literature about educational timetabling is reviewed in Section 1.4, to introduce the current state of the art for different versions of educational timetabling problems. A summary of heuristic methods to solve educational timetabling problems is given in Section 1.4.1. Section 1.4.2 contains an overview of exact methods used to solve different timetabling instances.

Section 2 deals with formalizing the CAMPUS 02 timetabling problem, informally introduced in Section 1.1. In Section 2.1 the CAMPUS 02 timetabling problem is shown to be a special case of the general timetabling problem from Section 1.3. A linear integer programming formulation is given in Section 2.2.

The theoretical foundations of educational timetabling are introduced in Section 3. The connection between timetabling and graph coloring and relevant extensions to classical graph coloring are the content of Section 3.1. In Section 3.2 the computational complexity of educational timetabling problems is reviewed.

The main part of the thesis is Section 4. It contains the development of heuristics for the CAMPUS 02 timetabling problem. Section 4.1 explains the basic building parts used by the heuristic methods developed. In Section 4.2 these are combined to a complete heuristic. Section 4.3 introduces hyper-heuristics for the CAMPUS 02 timetabling problem. This includes an introduction to genetic algorithms and based on that the development of a selection hyper-heuristic. This selection hyper-heuristic is also generalized to a multi-objective formulation of the problem.

Section 5 summarizes the computational results for the CAMPUS 02 timetabling problem using the methods introduced in this thesis. The results obtained by trying to solve the problem using a general purpose integer programming solver are summarized in Section 5.1. A summary of the results obtained by these heuristics is given in Section 5.2.

Section 6 contains a short conclusion of the results obtained by the methods in this thesis. Based on that an outlook about further possible research directions is given.

1.3. The Timetabling Problem and its Variants in Education

Burke et al. [27] define a generic timetabling problem. In the following we introduce this general timetabling problem and define several well known variants of school and university timetabling as special cases. The CAMPUS 02 timetabling problem, informally explained in Section 1.1, is also a special case. We use the terminology introduced in this section throughout the thesis.

Definition 1.1 (General timetabling problem, Burke et al. [27])**.** A timetabling problem is given by the following input:

- A finite set of times T, which are available for scheduling.
- A finite set of resources R.
- A finite set of meetings M, which are fixed collections of time slots and resource slots.
- A finite set of constraints C.

A time slot is a variable that can be assigned with a time in T and a resource slot is a variable that can be assigned with a resource in R.

The problem asks to find an assignment of times and resources to the slots inside meetings, such that all the constraints in C are fulfilled. Additionally the constraints can be split into hard and soft constraints, where hard constraints must be fulfilled and soft constraints should be fulfilled as far as possible.

Note that in some practical examples such an assignment must not assign values to all the time and resource slots.

Remark. As shown in many examples of timetabling problems, the resource slots are often preassigned in all meetings and only the time slots have to be scheduled. The resources can be used to model lecturers or classes.

The meetings often correspond to courses in practical timetabling problems. For instance one meeting can contain a resource slot preassigned with a class, another resource slot preassigned with a lecturer and the time slots in this meeting correspond to all lessons the teacher has to teach in this class.

The following is a list of constraints that are common among (almost all) timetabling problems.

Completeness constraint Every (time or resource) slot has to be assigned by a value.

No-clash constraint A resource must not be assigned to resource slots in any pair of meetings sharing a time slot assigned with the same time.

Availability constraint For each resource $r \in R$ there are subsets $T_r \subseteq T$ of time defined at which the resource is available. This constraint is violated if in a meeting in which the resource r is assigned to a resource slot, a time slot exists with a time $t \notin T_r$ assigned.

Resource type constraint For a resource slot s only a subset $R_s \subseteq R$ of resources is assignable.

No idle-time constraint For a resource $r \in R$ and for given time periods (for instance days) there should be no unplanned time units between blocks of times assigned to meetings the resource is assigned to.

Remark. If resource slots are not preassigned, they almost always have a resource type constraint associated with them. For example such a slot could be reserved for a lecturer, who is able to teach a course and only lecturers who can teach this course are feasible assignments.

1.3.1. The Class-Teacher Timetabling Problem

Basic class-teacher timetabling (Gotlieb [33]) The basic class-teacher timetabling problem is a timetabling problem where teachers and classes are resources and every meeting consists of one preassigned teacher resource slot, one preassigned class resource slot and one time slot. For all the time slots there are completeness constraints. The no-clash constraints are added as hard-constraints for all resources.

Remark. Instead of modeling courses by multiple meetings with one time slot, we could also use one meeting for each pair of a class and a teacher, containing one time slot for each meeting before. This version is sometimes better suited for generalizations.

This basic version of the problem has a strong connection to the edge-coloring problem in a bipartite graph. More details about this and the consequences for computational complexity are shown in Section 3.

To make this model applicable to practical timetabling problems there exist several extensions to this basic formulation. We list some common possibilities based on Burke et al. [27], Pillay [47] and Post et al. [53].

- Teachers are not always available.
- Time slots inside meetings can be forced to be assigned with contiguous times.
- Some meetings can include several class resource slots, meaning that a course is taught for several classes at once.
- There can be a room assignment modeled by adding rooms as resources. This is for instance necessary to model the use of some special rooms (e.g. computer rooms) for some courses.

In Section 2.1 it is shown that the CAMPUS 02 timetabling problem is a special version of the class-teacher timetabling problem.

Class-teacher timetabling problems are often also called *school timetabling* problems, because this is the common setting in most elementary schools. The *Third International Timetabling Competation (ITC2011)* was concerned with high school timetabling problems. An XML standard called *XHSST* for describing different practical high school timetabling problems is introduced by Post et al. [53]. This standard allows the formulation of very general timetabling problems. Solution approaches for class-teacher timetabling cannot be used to tackle these general problems, but many practical benchmark instances from the ITC2011 can still be seen as generalizations of the class-teacher timetabling problem.

1.3.2. University Timetabling Problems

The university course timetabling problem takes into account that courses are not taught for a single fixed class of students but are selected by students. There are two different variants of this problem.

Post enrollment course timetabling The post enrollment timetabling problem considers the planning phase after students have registered for their fixed courses. It is modeled as a special case of the general timetabling problem (Definition 1.1) by defining the set of students as resources. These students are then preassigned to resource slots in the meetings corresponding to courses they are enrolled to.

Curriculum based course timetabling The curriculum based course timetabling problem considers a fixed curriculum which consists of sets of courses that are not allowed to be scheduled at the same time. This is an extension of class-teacher timetabling because courses can be taught for multiple classes (called curricula in this context). This can be handled in the setting of the general timetabling problem (Definition 1.1) by having multiple resource slots, preassigned with each of the classes a course is assigned to, in the meetings of each course.

These problems can be extended in several ways, similarly to the class-teacher timetabling problem. In this setting, room assignment is often part of the problem, where the room slots are not preassigned.

For the *Second International Timetabling Competition (ITC2007)* very general versions of these problems are introduced, to create standardized benchmark instances for research on university timetabling. For the curriculum based course timetabling problem Bonutti et al. [6] define a problem based on the timetabling problem at the University of Udine. There are many benchmark instances available for this problem and a lot of researchers work on this problem (see Section 1.4). It is known as the *Udine problem* in literature.

Similarly for the post enrollment course timetabling problem Lewis et al. [38] define a problem version, which is a generalization of the problem used for the *First International Timetabling Competation (ITC2002).*

At universities exams are often scheduled independent from the course scheduling. This problem is formalized as the *University examination timetabling* problem. The basic structure of the university examination timetabling problem is similar to the university course timetabling problem.

University examination timetabling The set of meetings consists of the exams to be scheduled. Each meeting contains several time slots for assignment and contains preassigned resource slots for students that need to take the exam. In addition the meetings contain resource slots for rooms that are needed for the exam and lecturers that are necessary for supervision.

The major difference to the course timetabling problem are the constraints. For instance with courses it is often considered as good, if there is no idle time in between for the students, where on the contrary blocks of exams are considered as bad by students.

At the Second International Timetabling Competition (ITC2007) a standardized version of the university exam timetabling problem was proposed by McCollumn et al. [42].

1.4. Literature Review on Solution Methods for Educational Timetabling

In the Operations Research literature there is no single standard timetabling problem that is studied, but there are lots of different similar versions of practical timetabling problems analyzed and solved in different publications. Most of these problems can be classified as special versions of the general timetabling problem introduced in Section 1.3. As mentioned in Section 1.3.2 for the ITC2007 very general standardized versions of the university course timetabling problem and the university exam timetabling problem are introduced. Many publications try to tackle these problems, because this allows researchers to compare their results with previous results.

Because of this vast quantity of literature on educational timetabling it is not possible to cover all the publications and research directions. We try to direct the reader to survey articles showing the development and current state of the art on techniques and theory relevant to this thesis.

Section 3.2 contains an overview of the computational complexity of educational timetabling problems, showing that most variants of timetabling problems are NP-complete. That is why a lot of research is done to develop heuristics for timetabling problems. Section 1.4.1 reviews the current literature on this topic. In Section 1.4.2 different exact methods to solve timetabling problems with their respective objective functions are shown.

To keep the review sections short we do not give formal definitions of all terms used. Instead we refer the reader to the cited surveys and papers for full definitions and explanations. If these terms are used in later sections to describe the approach used in this thesis, they are defined as needed.

1.4.1. Heuristic Methods

The early literature about educational timetabling heuristics is summarized in Carter's survey from 1986 [14]. It is observable that the connection between timetabling and graph coloring is already used in these heuristic methods. This connection is first mentioned by Welsh and Powell [60]. Papers from this period try to develop heuristics for single practical timetabling problems. These heuristics are shown to produce good timetables. The heuristics are specialized to these problems and are not tested for other instances or different variants of the timetabling problem at other institutions. This makes these heuristics not applicable to other university timetabling problems in practice.

During the nineties researchers worked on developing more general timetabling heuristics by applying meta-heuristics such as genetic algorithms, simulated annealing and tabu search. Burke et al. [10] wrote a review of the state of the art on university timetabling in 1997. This development is also described in a survey from Carter and Laporte [43].

Because of the success in applying tailored heuristics and meta-heuristics to different timetabling problems recent research tries to develop general methods to solve arbitrary similar timetabling problems for different institutions. Meta-heuristics are a first attempt to use more general algorithms that can be used for different types of similar timetabling problems. But they were still adapted to the concrete problem instances. Current research tries to develop methods that are suited to automatically generate heuristics for new instances of educational timetabling problems and these methods are called hyper-heuristics.

Hyper-Heuristics

The term hyper-heuristic is relatively new and was first used in 2000 in a paper by Cowling et al. [18] for personnel scheduling. The survey article by Qu et al. [54] from 2009 summarizes the first applications of hyper-heuristics to university timetabling problems in the 2000s. In 2010 Burke et al. [9] present a classification and definition of hyper-heuristics, covering the previous initial work on the topic. They define a hyper-heuristic as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". The classification given in their paper is shown in Figure 1.1. It is a classification according to two dimensions: (i) the nature of the heuristics search space and (ii) the source of feedback. Burke et al. [11] present the state of the art on hyper-heuristics in their survey article. They also include a survey on the applications of hyper-heuristics to university timetabling problems. It is shown that by these methods good results for different timetabling problems can be obtained using the same algorithm.

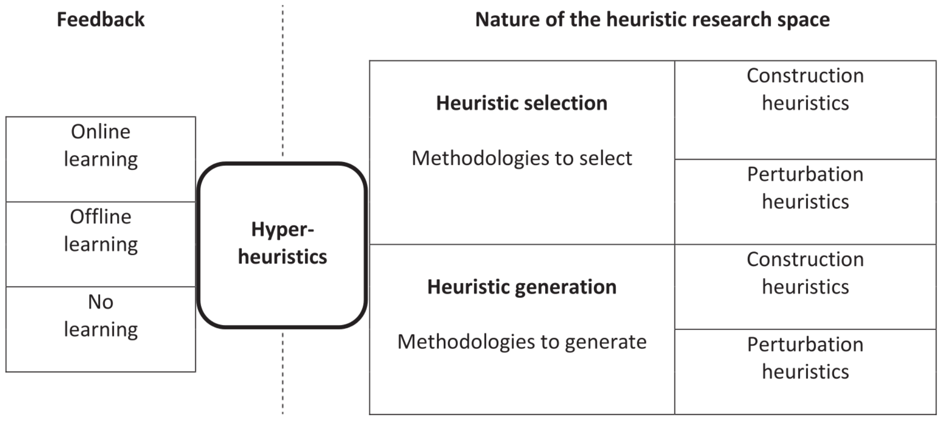


Figure 1.1.: Classification of hyper-heuristics. Source: Burke et al. [9, p. 453].

We want to note that most of the hyper-heuristics for educational timetabling can be classified as selection heuristics according to Figure 1.1. There is some research on heuristic generation of constructive heuristics for university timetabling, for instance Asmuni et al. [2] and Pillay [48].

For class-teacher timetabling there are less publications using hyper-heuristics to solve the problem. Reasons are that some of these problems are smaller and can therefore be solved by exact methods. Another reason might be that there was less research on these problem instances because no track for class-teacher timetabling existed for the ITC2007. Pillay wrote a survey [50] in which the author notes that research in this direction is missing. An investigation into applying different hyper-heuristics known from university timetabling to the class-teacher timetabling problem is also published in [50]. A constructive selection and a generation hyper-heuristic and a selection hyper-heuristic using perturbation heuristics are applied to the problem. This is especially interesting for the CAMPUS 02 timetabling problem, because it can be formulated as a special instance of the class-teacher timetabling problem (see Section 2.1). Currently we observe an increase in the research on class-teacher timetabling because the topic of the ITC2011 was school timetabling [52]. New publications on this subject [1] often use the constructive heuristic implemented in the KHE14 software from Kingston [34] to generate an initial solution and then apply perturbation hyper-heuristics to calculate better solutions.

We know of no publications that generate perturbation heuristics using hyper-heuristics for educational timetabling problems. But Burke et al. [8] show that good local search heuristics for bin packing can be generated by hyper-heuristics. Research on this subject for educational timetabling would be of interest, to further decrease the need of human interaction in the timetabling process.

Multi-Objective Methods

It is already mentioned that the objective function of educational timetabling problems is not clear. Most authors minimize a (weighted) sum of unsatisfied soft constraints. In practice these soft-constraints correspond to wishes of different parties (students, lecturers, other administration). Some research is dedicated to formulating the problem as a multi-objective optimization problem, by splitting the objectives for different stakeholders. Burke et al. [12] analyze the multi-objectivity of the exam timetabling problem of the ITC2007 in this way, obtaining a bi-objective optimization problem with one objective for students and the other for administration. This shows that there is a need for extensions to some early work of Burke et al. [7].

Zitzler proposed in his PhD-thesis [61] an evolutionary algorithm to solve multi-objective optimization problems. Generalizations of this approach are often applied to solve hard multi-objective optimization problems. An example of an application to educational timetabling is Datta et al. [20].

In Silva et al. [56] applications of meta-heuristics to timetabling problems are reviewed. It is mentioned that it is not practical to view all objectives of different stakeholders as different objective functions, because this would lead to too many objectives (every lecturer, student, class, course has its own objective), and again some weighted average for each class of stakeholders is often used as described in Burke et al. [12]. The development of many objective optimization techniques to tackle this problem would be of interest.

Current research in the field again tries to develop hyper-heuristics for multi-objective methods to reduce the need for human interaction when applying methods to new similar problems. Most of this research is conducted after 2010, where Maashi et al. [39, 40] give two examples of creating a selecting hyper-heuristic, working with different multi objective evolutionary algorithms as its meta-heuristics. We know of no application of such techniques to educational timetabling.

1.4.2. Exact Methods

The drawback of heuristic methods shown above is that we cannot prove optimality of solutions and also do not know of any bounds to the optimal solutions. This is why even knowing that heuristics have shown to produce good results there is an interest in developing exact methods for educational timetabling problems. Beginning with the first publications on timetabling the problem was formulated as an integer linear program, in most cases using binary variables. These problems are in general hard to solve but using modern solvers and techniques on state of the art computers some educational timetabling problems can be solved exactly.

The survey article on educational timetabling by Kristiansen et al. [36] contains a review on the development of exact methods for the course timetabling problem in [36, Section 3.3.5] and on exam timetabling in [36, Section 5.3.6]. Pillay [50] also reviews the integer programming methods applied to school timetabling in [50, Section 3.7].

In literature there are different approaches using exact integer programming formulations.

Some articles present new formulations of the problem as an integer program and solve these directly using state of the art solvers. Often these formulations are just for some specific problem or are improved versions that are easier to solve by general purpose solvers. All these formulations have in common that they use 0-1-variables for modeling. We know of only one interesting exception, where variables represent the assignment of complete teaching schedules, from McClure and Wells [41].

Other articles include research on specialized cuts for the problems, studied to develop tailored branch and cut methods like Santos et al. [55]. There is also research on column generation for educational timetabling problems, for instance Papoutsis et al. [46].

Also lower bounds are derived, which can be used to improve the performance of integer programming solvers, like Cacchiani et al. [13].

Another different approach is to partition the problem into multiple easier stages and solve each of these exactly using integer programming, for instance Sørensen et al. [58].

2. The CAMPUS 02 Timetabling Problem

2.1. Definition based on the General Timetabling Problem

The CAMPUS 02 timetabling problem is formalized as a general timetabling problem, using the notation introduced in Defintion 1.1.

2.1.1. Input Parameters

The following parameters are given for each instance of the CAMPUS 02 timetabling problem:

- D the ordered set of days to schedule. We use the convention that for $d \in D$ $\text{ord}(d) \in \{1, \dots, |D|\}$ gives the ordered index of d.
- $H := \{1, 2, \dots, 9\}$ the set of hours available at each day.
- L the set of lecturers.
- K the set of classes.
- C the set of courses.
- $CK \subseteq C \times K$ the curriculum.
- $CL \subseteq C \times L$ an assignment of lecturers to classes (some classes are split into multiple groups that are taught simultaneously by multiple lecturers)
- $DK \subseteq D \times K$ an assignment of days to classes (days at which a class is available). Courses taught for a class can only be scheduled on the days the class is available according to this set.
- $WE \subseteq D \times D$ a set explaining which pairs of days correspond to weekends. Of course it holds, that each day $d \in D$ is only contained in a single weekend in WE. The first entry in each tuple corresponds to the Friday, the second entry to the Saturday.

- $W \subseteq DK$ the subset of days at which a class has its intensive week. These days correspond to Monday to Thursday of one real week and it therefore always holds that for every class $k \in K$ there is either no intensive week or

$$|\{d\colon (d,k) \in W\}| = 4.$$

  Note that in the CAMPUS 02 timetabling problem all days in D are either part of some weekend or contained in the intensive week of some class.

- $\mathit{NOGO} \subseteq D \times L$ the NoGo-days of lecturers.
- $\mathit{CourseHours}(c) \in \mathbb{N}_0$ for each course $c \in C$ defines the number of hours to schedule for c.
- $\mathit{ExamHours}(c) \in \mathbb{N}_0$ for each course $c \in C$ defines the number of hours to schedule for the exam of c.
- $l(c) \in \mathbb{N}$ for each course $c \in C$ defines the lower block size for course c.
- $u(c) \in \mathbb{N}$ for each course $c \in C$ defines the upper block size for course c.

  A block is the number of consecutive hours on a day at which a course is scheduled.

An example of a practical input at CAMPUS 02 University of Applied Sciences is given in Appendix B.

2.1.2. Problem Definition

We define the corresponding timetabling problem using the general timetabling problem from Definition 1.1 by setting

- the set of possible times for assignment to time slots, $T := \{(d,h)\colon d \in D, h \in H\}$,
- the resources assigned to meetings, which are in this case classes and lecturers, $R := K \,\dot\cup\, L$,
- $M := \{m(c)\colon c \in C\}$ where $m(c)$ is the meeting for course c defined by

$$\begin{aligned} m(c) := (t_i, e_j, l, k)\colon\ & i = 1, \dots, \mathit{CourseHours}(c) \\ & j = 1, \dots, \mathit{ExamHours}(c) \\ & l_n = l' \ \forall (c,l') \in CL \\ & k_n = k' \ \forall (c,k') \in CK\}, \end{aligned}$$

  with t_i, e_j the unassigned time slots and l, k the preassigned resource slots for the course c.

We refer to the t_i with time slot and to the e_i with exam time slot.

For the CAMPUS 02 timetabling problem, there is only one class slot k in each meeting, so $|(c,k')\colon (c,k') \in CK| = 1$ for all $c \in C$.

In addition a set of times is called a block, if all the times are on the same day and the corresponding hours are consecutive. That means for a day d and a starting hour t_1 and end hour t_2 the set

$$B = \{(d,t)\colon t_1 \le t \le t_2\}$$

is the corresponding block of times.

If the times in B are assigned to the meeting $m(c)$ we say that the course c is scheduled at the block B.

In the following we state the constraints:

- Completeness constraints for all time slots t_i and exam time slots e_i in all meetings.

  That means all the time slots need to be assigned with a time $t \in T$. This leads to the fact that for each course $c \in C$ exactly $\mathit{CourseHours}(c)$ hours of the course are scheduled and $\mathit{ExamHours}(c)$ hours are scheduled for the exam.

- No-clash constraints for all resources in all meetings.

  These constraints enforce that at any time $t \in T$ only one course is scheduled for each class and lecturer.

- Availability constraints for all classes $k \in K$, where the availability sets are

  $$T_k := \{(d,h) \in T\colon (d,k) \in DK, h \in H\}.$$

  This enforces that courses for classes are only scheduled on days at which the class is available.

- Availability constraints for lecturers $l \in L$, where the availability sets are given by

  $$T_l := \{(d,h) \in T\colon (d,l) \notin \mathit{NOGO}, h \in H\}.$$

  Courses are only scheduled at times such that the assigned lecturers are available.

- Constraints to enforce blocking:
  - At every day $d \in D$ if course $c \in C$ is scheduled, it has to be scheduled as a single block of length $l(c) \le n \le u(c)$, meaning that in $m(c)$ the course starts at hour $h \in H$ and we set $t_{i_k} := (d, h+k)$ for $k = 0, \dots, n-1$ with slot indices i_k.

    We call such feasible blocks of time assigned to $m(c)$ the *time blocks* of course c.

- No idle time constraints for all the classes $k \in K$.
- Constraints for exams:
  - The exam of a course starts at least one week after the last block of the course. That means for every course $c \in C$ in the meeting $m(c)$
  $$\max_{t_i=(d,h)} \operatorname{ord}(d) \leq 2 + \min_{e_i=(d,h)} \operatorname{ord}(d).$$
  - There are no exams on days during the week, meaning $\forall m(c) \in M, (d,k) \in W, h \in H : e_i \neq (d,h)$.
  - Exams have to be scheduled as one block of length $\mathit{ExamHours}(c)$ on a single day d starting at some hour $h \in H$, meaning $e_{i_k} := (d, h+k)$ for $k = 0, \dots, \mathit{ExamHours}(c) - 1$ with slot indices i_k.
  - Exams have to start during the first 3 hours of each day.
- Lecturers are not allowed to teach the last time on Friday and the first time on Saturday during each weekend.
- All days $(d,k) \in W$ in the intensive week must be nearly fully planed, meaning there are at least 8 hours planned at this day.

The list above contains all the hard constraints for the problem. Additional soft constraints that should be fulfilled if possible are listed below:

- The number of weekends at which there are courses schedule should be minimized.
- If a day for a class $(d,k) \in DK$ is used for scheduling in a meeting $m(c)$ for a course $(c,k) \in CK$ this day should be as full as possible for the class k.
- For a class k the number of consecutive free weekends should not exceed two.
- On all days there should be no idle times for all lecturers.
- Hard lectures (defined by some hardness parameter) should start at early times.
- Exams should be planned on Fridays.

Remark. If we would additionally enforce that all courses are only taught by one single lecturer, so $|(c,l') \colon (c,l') \in CL| = 1$ for fixed c, the problem would be an instance of the class-teacher timetabling problem, with special constraints.

Based on this formal problem definition it is possible to define a timetable for the CAMPUS 02 timetabling problem.

Definition 2.1. We call an assignment of times to the time slots *timetable*, if all constraints, except the completeness constraints, are satisfied. If in addition all completeness constraints are fulfilled the assignment is called *complete timetable.* Else it is called *partial timetable.*

If the assignment does not satisfy other constraints than the completeness constraints, it is called *infeasible.*

This definition includes the assignment, where only the preassigned time slots have times assigned as a partial timetable.

If a time $(d, h) \in T$ is assigned to a meeting $m(c)$ of the course $c \in C$ we say, that the course c is scheduled at (d, h). Because of the blocking constraints for courses, it is also possible to refer to the block of course c on day d, referring to the times (d, h) assigned to $m(c)$. Because of the no idle time constraints for classes analogously one can refer to the block of class k on day d, which corresponds to the times

$$\{(d, h) \in T \colon \exists (c, k) \in CK \colon (d, h) \in m(c), k \in m(c)\}.$$

Remark. Note that we write $r \in m$, if the resource r is assigned to a resource slot in the meeting m and $t \in m$, if the time t is assigned to a time slot in meeting m.

2.1.3. Objective Function

Which of the feasible timetables are considered as *good* is not that clear. This is a problem quite common among educational timetabling problems. It is not clear how to formally compare different timetables. The easiest and most used way is to assign an objective value to each timetable using a weighted sum of unsatisfied soft constraints. We also use this approach for the integer programming formulation and the heuristics.

For later reference, functions corresponding to the most important soft constraint violations are defined.

Free weekends For each class k the function $\text{freeWeekends} : K \to \mathbb{N}_0$ counts the number of weekends, for which no time is assigned to a meeting containing the class k. It is of course desirable to maximize this function for each class. Analogously we can define a function $\text{usedWeekends} : K \to \mathbb{N}_0$ counting the number of weekends used for scheduling.

Short days For each class k the function $\text{shortDays} : K \to \mathbb{N}_0$ counts the number of days, for which the block of times on that day, assigned to meetings containing the class k, has a length smaller than 8. It is not considered good for students, to have such short days in their timetable. This number should therefore be tried to minimize for each class k.

Unused Hours For each class k the function$\text{unusedHours} : K \to \mathbb{N}_0$ counts for each day at which a block of times is assigned to a meeting containing the class k the number of hours on this day that are not assigned to any meeting containing the class k.

Free Periods For each class k the function freePeriods : $K \to \mathbb{N}_0$ counts the number of free days between two days which are assigned to meetings of k, that exceed two (one free weekend in between does not count). These long periods of no teaching are considered bad for the students and should be minimized.

Days used For a lecturer $l \in L$ the function daysUsed : $L \to \mathbb{N}_0$ counts the number of days at which a meeting containing the lecturer is scheduled.

Idle Times For a lecturer $l \in L$ the function idleTimes : $L \to \mathbb{N}_0$ counts the number of hours not assigned to any meeting containing l, between two meetings with times on the same day, containing the lecturer l. Such idle times should be minimized for each lecturer.

Late meetings For a course $c \in C$ the function lateMeetings : $C \to \mathbb{N}_0$ counts the number of times (d, h) with $h > 6$ assigned to $m(c)$. These late meetings should be minimized for hard courses.

Saturday exams For a class $k \in K$ the function saExams : $K \to \mathbb{N}_0$ counts the number of exams that are scheduled on Saturdays for the class k.

Looking at the different soft constraints in more detail, it is observable that they represent wishes and objectives of different (competing) parties. In the case of the CAMPUS 02 timetabling problem, these parties are the resources (lecturers and classes) and the courses. This leads to a multi-objective problem with objectives for each party. Section 1.4.1 contains a review of the literature about methods to solve this kind of formulation for similar problems. A hyper-heuristic to solve the multi-objective formulation of the CAMPUS 02 timetabling problem is developed in Section 4.3.3.

2.2. Modeling as an Integer Program

The CAMPUS 02 timetabling problem from Section 2.1 is modeled as an integer linear program (IP) using binary variables. This gives another formalization of the problem and allows the solution of the problem using integer programming solvers. We follow an approach similar to the ones reviewed in Section 1.4.2.

The model is based on the formulation in Section 2.1.

2.2.1. Variables

Binary variables are introduced for each possible time $t \in T$ combined with every course $c \in C$, that indicate if time t is assigned to a time slot in meeting $m(c)$. These variables are denoted by $\mathbf{x}_{d,h,c}$ for $t = (d, h)$. If $\mathbf{x}_{d,h,c} = 1$ the time (d, h) is assigned to a time slot in $m(c)$ and $\mathbf{x}_{d,h,c} = 0$ otherwise.

Analogously variables $\mathbf{e}_{d,h,c}$ are defined to indicate, if a time $t = (d, h)$ is assigned to an exam time slot in meeting $m(c)$. An assignment $\mathbf{e}_{d,h,c} = 1$ indicates that the time (d, h) is assigned to an exam time slot in $m(c)$ and $\mathbf{e}_{d,h,c} = 0$ otherwise.

These variables are sufficient to encode a complete solution of the problem. But we need to introduce further variables to model the constraints and objectives of the problem using linear (in)equalities and functions.

One major reason leading to more variables are the blocking constraints. These force us to introduce additional variables for each time $t = (d, h)$ and each block element (course or class) indicating the start of a block. These variables are denoted with $\mathbf{s}_{d,h,c}$ for the time slots and $\mathbf{es}_{d,h,c}$ for the exam time slots. The blocking constraints introduced in the next section enforce that $\mathbf{s}_{d,h,c}$ and $\mathbf{es}_{d,h,c}$ are equal to 1 if and only if a block of times is assigned to the time slots or exam time slots of $m(c)$ that starts at time (d, h) and 0 otherwise.

Additional variables are defined as needed next to the corresponding constraints or objectives.

2.2.2. Hard Constraints

Linear equations and inequalities are used to model the constraints of the problem. For some of the constraints additional variables are needed. The set of constraints defined in this Section gives a characterization of all timetables that form a feasible solution to the problem.

- Completeness constraints for time slots and exam time slots.

$$\sum_{(d,h)\in D\times H} \mathbf{x}_{d,h,c} = \mathit{CourseHours}(c) \qquad \forall c \in C$$
$$\sum_{(d,h)\in D\times H} \mathbf{e}_{d,h,c} = \mathit{ExamHours}(c) \qquad \forall c \in C$$

- No-clash constraints for lecturers and classes.

$$\sum_{c\colon (c,l)\in CL} (\mathbf{x}_{d,h,c} + \mathbf{e}_{d,h,c}) \le 1 \qquad \forall l \in L, (d,h) \in T$$
$$\sum_{c\colon (c,k)\in CK} (\mathbf{x}_{d,h,c} + \mathbf{e}_{d,h,c}) \le 1 \qquad \forall k \in K, (d,h) \in T$$

- Availability constraints for classes.

$$\mathbf{x}_{d,h,c} = 0 \qquad \forall (d,k) \notin DK, c \in C\colon (c,k) \in CK, h \in H$$
$$\mathbf{e}_{d,h,c} = 0 \qquad \forall (d,k) \notin DK, c \in C\colon (c,k) \in CK, h \in H$$

- Availability constraints for lecturers

$$\mathbf{x}_{d,h,c} = 0 \qquad \forall (d,l) \in \mathit{NOGO}, h \in H, c \in C\colon (c,l) \in CL$$
$$\mathbf{e}_{d,h,c} = 0 \qquad \forall (d,l) \in \mathit{NOGO}, h \in H, c \in C\colon (c,l) \in CL$$

- Constraints that ensure exams are correctly scheduled:
  - The exam for course c has to start at least one week after the last time assigned to a time slot of $m(c)$.

$$\sum_{\substack{h' \in H, d' \in D\colon \\ \mathrm{ord}(d') \geq \mathrm{ord}(\mathrm{prev}(d))}} \mathbf{x}_{d',h',c} \leq \mathit{CourseHours}(c)(1 - \mathbf{es}_{d,h,c}) \qquad \forall c \in C, d \in D, h \in H$$

  - No exams are scheduled on days of the intensive week W.

$$\mathbf{e}_{d,h,c} = 0 \qquad \forall (d,k) \in W, h \in H, c\colon (c,k) \in CK$$

  - For each class there is only one exam allowed during each weekend.

$$\sum_{\substack{c\colon (c,k) \in CK \\ h \in H}} (\mathbf{es}_{d_1,h,c} + \mathbf{es}_{d_2,h,c}) \leq 1 \quad \forall (d_1, d_2) \in WE, k \in K$$

  - Exams must start during the early hours of a day.

$$\mathbf{e}_{d,h,c} = 0 \quad \forall d \in D, c \in C, h \in H\colon h > 3$$

- The intensive week has to be nearly fully scheduled.

$$\sum_{\substack{h \in H \\ c \in C\colon (c,k) \in CK}} (\mathbf{x}_{d,h,c} + \mathbf{e}_{d,h,c}) \geq 8 \qquad \forall (d,k) \in W$$

- Lecturers are not allowed to teach the last hour on Friday and the first hour on the next Saturday.

$$\sum_{c\colon (c,l) \in CL} (\mathbf{x}_{d_1,\mathrm{last}(H),c} + \mathbf{e}_{d_2,\mathrm{first}(H),c}) \leq 1 \qquad \forall l \in L, (d_1, d_2) \in WE$$

- Blocking constraints:
  - Blocking for single courses on each day.

    To formulate this as a linear constraint we need to introduce variables $\mathbf{s}_{d,h,c}$ for normal courses and $\mathbf{es}_{d,h,c}$ for exams, that indicate starts of blocks.

    There exists only one block for each course on each day and only one exam block at all, which is enforced by the following constraints.

$$\sum_{h \in H} \mathbf{s}_{d,h,c} \leq 1 \qquad \forall c \in C, d \in D$$

$$\sum_{d \in D, h \in H} \mathbf{es}_{d,h,c} \leq 1 \qquad \forall c \in C$$

The following inequalities enforce the correct block lengths.

$$\left(\sum_{h\in H} \mathbf{s}_{d,h,c}\right) l(c) \le \sum_{h\in H} \mathbf{x}_{d,h,c} \qquad \le \left(\sum_{h\in H} \mathbf{s}_{d,h,c}\right) u(c) \quad \forall c \in C, d \in D$$

We need to enforce, that the variables $\mathbf{s}$ and $\mathbf{es}$ force the variables $\mathbf{x}$ and $\mathbf{e}$ to build consecutive blocks on each day d, for each course c. The correctness of the following constraints is based on Lemma 2.1.

$$\mathbf{x}_{d,h,c} \begin{cases} = \mathbf{s}_{d,h,c} & \text{if } h = 1 \\ \le \mathbf{s}_{d,h,c} + \mathbf{x}_{d,h-1,c} & \text{else} \end{cases} \quad \forall c \in C, d \in D, h \in H$$

$$\mathbf{e}_{d,h,c} \begin{cases} = \mathbf{es}_{d,h,c} & \text{if } h = 1 \\ \le \mathbf{es}_{d,h,c} + \mathbf{e}_{d,h-1,c} & \text{else} \end{cases} \quad \forall c \in C, d \in D, h \in H$$

– Additionally we need that the courses taught for one class on each day form a block. This can be done analogously introducing block start variables $\mathbf{ds}_{d,h,k}$ for each time $(d,h) \in T$ and class $k \in K$. Using these we enforce blocking as for courses.

$$\sum_{h\in H} \mathbf{ds}_{d,h,k} \le 1 \quad \forall k \in C, d \in D$$

$$\sum_{c\colon (c,k)\in CK} (\mathbf{x}_{d,h,c} + \mathbf{e}_{d,h,c}) = \begin{cases} = \mathbf{ds}_{d,h,k} & \text{if } h = 1 \\ \le \mathbf{ds}_{d,h,k} + \\ \quad \sum_{c\colon (c,k)\in CK} (\mathbf{x}_{d,h-1,c} + \mathbf{e}_{d,h-1,c}) & \text{else} \end{cases}$$
$$\forall k \in K, d \in D, h \in H$$

The following lemma proves the correctness of the blocking constraints.

Lemma 2.1. *Given some index set $I = \{1, 2, \dots, n\}$, two vectors of variables $\mathbf{x}_i$ and $\mathbf{s}_i$ for $i \in I$ the constraints*

$$\mathbf{x}_i \begin{cases} = \mathbf{s}_i & \text{if } i = 1 \\ \le \mathbf{s}_i + \mathbf{x}_{i-1} & \text{else} \end{cases} \quad \forall i \in I$$

enforce $\mathbf{s}_i = 1$ whenever $\mathbf{x}_i = 1$ and $\mathbf{x}_{i-1} = 0$ for $i > 1$, and $\mathbf{x}_0 = \mathbf{s}_0$.

It follows that whenever a new block starts in $\mathbf{x}$, the corresponding index in $\mathbf{s}$ must be set to 1. Now we can enforce a maximum of $k \in \mathbb{N}$ blocks by the constraint

$$\sum_{i\in I} \mathbf{s}_i \le k.$$

Proof. The case for $i = 1$ is trivial.

If $i > 0$, $\mathbf{x}_i = 1$ and $\mathbf{x}_{i-1} = 0$, then a new block starts and we have

$$1 = \mathbf{x}_i \leq \mathbf{s}_i + \mathbf{x}_{i-1} = \mathbf{s}_i.$$

$\square$

2.2.3. Objective Function and Soft Constraints

As already mentioned it is not clear how to define objective functions for timetabling problems. There are several wishes of different stakeholders, that are not hard constraints to the problem. It is shown how to count violations of these constraints using linear functions to allow incorporating them into the objective function of a linear integer program. One can then use different combinations of weighted sums of these to solve the problem with regard to different priorities. For some of these objectives it is again necessary to introduce new variables to the problem and control them with additional constraints.

- One important objective is to minimize the number of weekends used for each class $k \in K$. To formulate this as a linear constraint we introduce new variables $\mathbf{usedWeekend}_{(d_1,d_2),k} \in \{0,1\}$ for each weekend $(d_1,d_2) \in \mathit{WE}$. Using the following constraints we can enforce that $\mathit{usedWeekend}_{(d_1,d_2),k} = 1$ if a course of class k is scheduled during weekend (d_1,d_2).

  $$\sum_{h \in H} (\mathbf{ds}_{d_1 h,k} + \mathbf{ds}_{d_2,h,k}) \leq 2\mathbf{usedWeekend}_{(d_1,d_2),k} \quad \forall (d_1,d_2) \in \mathit{WE}, k \in K$$

  Using these variables we define linear functions for each class $k \in K$ counting the number of used weekends

  $$\text{usedWeekends}(k) := \sum_{(d_1,d_2) \in \mathit{WE}} \mathbf{usedWeekend}_{(d_1,d_2),k}.$$

- We would like to minimize the number of unused time slots on days, that are used for a class. This can be modeled by introducing variables $\mathbf{unusedHours}_{d,k} \in \mathbb{N}_0$ which we want to force to 0 if there are no courses for class k scheduled on day d and to the number of unused hours on day d else. We use the linear helper functions

  $$\begin{aligned} \text{dayUsed}(d,k) &:= \sum_{h \in H} \mathbf{ds}_{d,h,k} \\ \text{usedHours(d, k)} &:= \sum_{\substack{h \in H \\ c\colon (c,k) \in CK}} \mathbf{x}_{d,h,c} \end{aligned}$$

to define the necessary constraint.

$$\mathbf{unusedHours}_{d,k} \geq 9\,\mathit{dayUsed}(d,k) - \mathit{usedHours}(d,k) \quad \forall (d,k) \in DK$$

This allows **unusedHours** to take values larger or equal to the unused hours on a used day for a class and minimizing with respect to this value will force the variables to take the value needed. We want to point out that minimizing with respect to **usedWeekend**, also implicitly enforces a minimization of unused hours. This is why only one of these two options should be chosen, to keep the number of variables in the problem smaller.

- Lecturers wish to have a minimum number of blocks to teach. The number of blocks a lecturer teaches can be counted by introducing binary block start variables $\mathbf{ls}_{d,h,l} \in \{0,1\}$ for each lecturer $l \in L$. These can be forced to 1 for each block start using the technique from Lemma 2.1.

$$\sum_{c\colon (c,l)\in CL} (\mathbf{x}_{d,h,c} + \mathbf{e}_{d,h,c}) = \begin{cases} = \mathbf{ls}_{d,h,l} & \text{if } h = 1 \\ \leq \mathbf{ls}_{d,h,k} + & \\ \quad \sum_{c\colon (c,l)\in CL} (\mathbf{x}_{d,h-1,c} + \mathbf{e}_{d,h-1,c}) & \text{else} \end{cases}$$
$$\forall l \in L, d \in D, h \in H$$

  Now we can minimize with respect to these variables **ls**.

  Remark. It is possible that some lecturers do not like to teach big blocks and prefer to have multiple small ones. This can also be handled by maximizing with respect to **ls**.

- Exams should start as early as possible. If an exam for a fixed course $c \in C$ starts after some hour h_0, that can be easily determined using the exam start variables **es**.

$$\text{examLate}(c) := \sum_{d\in D, h>h_0} \mathbf{es}_{d,h,c}$$

- For hard courses it is preferred, that they start early. To measure the number of late starts for a course $c \in C$ we again define a analogous function.

$$\text{courseLate}(c) := \sum_{d\in D, h>h_0} \mathbf{s}_{d,h,c}$$

As shown in the literature review in Section 1.4, there exist different methods to define the objective function used to solve the problem based on these objectives. The most common approach is to use a weighted sum of these objectives and solve the problem using some weights as its input. After evaluating the solution the user can then modify these weights to improve some other aspects of the solution.

3. Theoretical Foundations of Educational Timetabling Problems

In this section the theoretical foundations of educational timetabling problems are introduced. In Section 3.1 the connections between timetabling problems and graph coloring problems are established. Section 3.2 contains an overview of computational complexity results about educational timetabling problems.

3.1. Connection to the Graph Coloring Problem

The well known connections between graph coloring problems and timetabling problems are described in this section. These will be used in later sections to obtain complexity results (see Section 3.2) and to develop efficient heuristics based on known graph coloring heuristics (see Section 4.1.1).

3.1.1. Basic Definitions and Results

We use the basic definitions and notations for graph theory as introduced in the standard reference by Diestel [26].

There are two different connected types of coloring problems in graphs known as the edge coloring and the vertex coloring problem.

Definition 3.1. Let $G = (V, E)$ be a graph. A function $c\colon E \to \mathbb{N}$ is called an *edge coloring* of G, if for every vertex $v \in V$ all edges adjacent to v are assigned a different value by c. We call these values colors.

The number of different colors used by an edge coloring is an interesting parameter and denoted by value(c). The chromatic index of G is defined as the minimum number of colors needed to obtain an edge coloring of G and denoted by

$$\chi'(G) := \min_{c \text{ edge coloring of } G} \text{value}(c).$$

Definition 3.2. Let $G = (V, E)$ be a graph. A function $c\colon V \to \mathbb{N}$ is called a *vertex coloring* of G, if for every edge $\{v, w\} \in E$ it holds that $c(v) \neq c(w)$. Again the values of c are called colors. This means that vertices that are connected by an edge in G must be colored using different colors. Analogously the number of colors needed by a vertex coloring is of interest and denoted by value(c). If a coloring c with value$(c) = n \in \mathbb{N}$ exists we call the graph n-colorable and the minimum number of colors needed by a vertex coloring is called the chromatic number and denoted by

$$\chi(G) := \min_{c \text{ vertex coloring of } G} \text{value}(c).$$

The corresponding optimization problems ask for a given graph G to find a coloring with minimum number of colors.

The first connection between a timetabling problem and coloring is established for the class-teacher timetabling problem and the edge coloring problem in a bipartite graph.

Theorem 3.1 (Csima [19], de Wera [21])**.** *The basic class teacher timetabling problem is equivalent to an edge coloring problem in a bipartite graph.*

Proof. Define the bipartite graph $G = (U \dot\cup V, E)$ corresponding to a given class-teacher timetabling problem. The set of teachers corresponds to the vertices in U and the set of classes corresponds to the vertices in V. We add an edge between a vertex $u \in U$ and $v \in V$ for each meeting that the teacher u has with the class v.

Now an edge coloring c in this graph assigns to each edge a value such that for all other edges adjacent to the same teacher and class this value is not used. Using this value as the time of the meeting corresponding to the edge gives a assignment of the time slots that satisfies the no-clash constraints for teachers and classes. So finding a minimum edge coloring gives an assignment of times to the meetings, using a minimum number of times. □

Definition 3.3. Let $G = (V, E)$ be a graph. The maximum degree in G is denoted by

$$\Delta(G) := \max_{v \in V} \deg(v).$$

For bipartite graphs there is a well known connection between the maximum degree and the chromatic index. If the referenced graph is clear, we just write $\Delta = \Delta(G)$.

Theorem 3.2 (König's theorem [35])**.** *Let G be a bipartite graph. Then $\chi'(G) = \Delta(G)$.*

In Section 3.2 the proof of Theorem 3.5 also includes a proof of Theorem 3.2 and shows that an minimal edge coloring in a bipartite graph can be calculated in polynomial time. In general graphs this problem is NP-complete, but an edge coloring with at most $\Delta(G) + 1$ colors can be calculated in polynomial time [26, Theorem 5.3.2; Vizing 1964].

In the following we introduce the well known conflict graph for the post-enrollment course timetabling problem. This concept is introduced because we define a similar conflict graph for the CAMPUS 02 timetabling problem in Section 3.1.3.

For the basic post-enrollment course timetabling problem we define this conflict graph and show how it can be used to obtain a connection between timetabling and coloring in a graph.

Definition 3.4. The *conflict graph* $G = (V, E)$ of a course timetabling problem has as its vertex set the set of all pairs of courses with its needed time slots

$$V := \{(c, t_i) : c \in C, t_i \in m(c)\}.$$

Because edges in this graph should show the severity of conflict if the time slots in two adjacent vertices are assigned the same time we introduce weights $w : C \times C \to \mathbb{N}_0$ between courses, defined by

$$w(c_1, c_2) := \begin{cases} \infty & \text{if } c_1 = c_2 \\ |\{s \text{ student} \colon s \in m(c_1), s \in m(c_2)\}| & \text{else.} \end{cases}$$

So for two different courses we count the number of students, that are enrolled in both courses. Now the edge set is given by each pair of vertices such that the corresponding courses have a positive weight

$$E := \{\{(c, t_i), (c', t_i')\} : w(c, c') > 0\}$$

and the weight function between courses can be extended to the edges by setting

$$w(\{(c, t_i), (c', t_i')\}) := w(c, c').$$

Remark. We can incorporate different conflicts (e.g. common lecturers) by adding additional edges with corresponding weights.

Now a vertex coloring can be interpreted as an assignment of times to the time slots in the vertices without conflicts.

3.1.2. Extensions of the Graph Coloring Problem

As already mentioned in Section 1.3 the basic timetabling problems are often extended by additional constraints in practice. This section is a review of methods, that allow to incorporate these constraints to coloring problems, or analyze more general coloring problems on graphs.

Neufeld and Tartar [45] show, that vertex coloring with preassigned vertices and prevention of colors for selected vertices can be reduced to general vertex coloring.

Theorem 3.3 (Neufeld & Tartar [45]). *Let $G = (V, E)$ be a graph and $P \subseteq V$ a subset of vertices with preassigned colors $c : P \to \mathbb{N}$. It is possible to extend c to a coloring of G using at most $n \in \mathbb{N}$ colors, if and only if the graph $G' = (V', E')$ with $V' = V$ and E' containing*

- *the edges E of G,*
- *edges joining vertices preassigned to different colors, that is*

$$\{\{v, w\} \colon v, w \in P, c(v) \neq c(w)\},$$

- *edges to neighbors of vertices preassigned to the same color (see Figure 3.1), that is*

$$\{\{u, w\} \colon u, v \in P, c(u) = c(v), \{v, w\} \in E\},$$

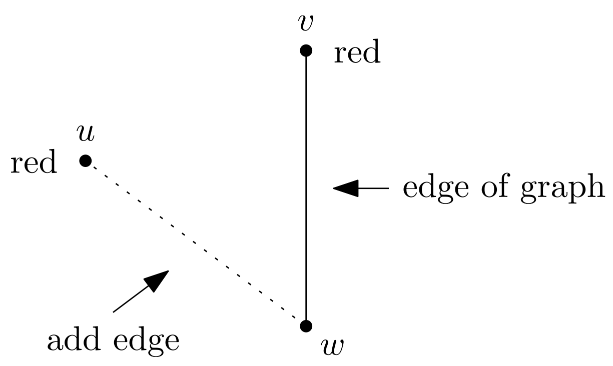


Figure 3.1.: Edges to neighbors of vertices preassigned to the same color are added.

is n-colorable.

Remark. Note that determining if an extension of the coloring as mentioned in Theorem 3.3 exists is NP-hard, because the graph coloring problem is NP-complete. But the theorem above shows that solving the graph coloring problem in the graph G' is sufficient for solving this problem.

Theorem 3.4 (Neufeld & Tartar [45])**.** *Let $G = (V, E)$ be a graph and $r : V \to 2^{[n]}$ a restriction function giving for each vertex $v \in V$ a set of colors which are not allowed to use for coloring v. The graph G is n-colorable with a coloring satisfying the restrictions given by r if and only if the graph $G' = (V', E')$ with vertex set*

$$V' := V \,\dot{\cup}\, \{v_i \colon i = 1, \dots, n\}$$

and edge set E' consisting of

- *the edges E of G,*
- *edges building a complete subgraph of the color vertices $\{\{v_i, v_j\} \colon i, j \in [n], i \neq j\}$,*
- *edges induced by the restriction function $\{\{w, v_i\} \colon w \in V, i \in r(w)\}$,*

is n-colorable.

Many other special constraints that lead to extensions of coloring are summarized in [22]. This also includes the enforcement of simultaneity, meaning that some edges (or vertices) are enforced to have the same color (regardless of feasibility of this decision). This allows the modeling of multiple groups for a course.

Another very common constraint, that is problematic for graph coloring, is the need for blocking of lectures. To cover these constraints a generalization of graph coloring, called interval coloring, is studied.

Definition 3.5. Let $G = (V, E)$ be a graph and $d : V \to \mathbb{N}$ a demand function. Then a function $c : V \to 2^{\mathbb{N}}$ is called an *interval coloring* of G, if it assigns an interval of integers (subsets of consecutive numbers), to each vertex such that for each vertex $v \in V$ the length of the interval fulfills $|c(v)| = d(v)$ and for each edge $\{u, v\} \in E$, $c(u) \cap c(v) = \emptyset$ holds.

Analogously for a demand function $d : E \to \mathbb{N}$ an interval edge coloring of G is defined as a function $c : E \to 2^{\mathbb{N}}$, such that an interval of integers is assigned and for every edge $e \in E$ it holds that $|c(e)| = d(e)$ and for two adjacent edges $e_1, e_2 \in E$ we have $c(e_1) \cap c(e_2) = \emptyset$.

Remark. This still does not always cover the requirements on blocking, because often blocks are restricted to be parts of days and a solution to the interval coloring problem cannot always be aligned in such a way, that this holds for all blocks in the solution. Nevertheless methods to solve these problems are of interest for applications to timetabling problems.

In [62] Čangalović and Schreuder develop an algorithm to calculate the interval chromatic number of a graph and apply it to a timetabling problem. Kubale [37] analyzes the interval chromatic number for several special graph classes and determines hardness results. It is also analyzed how to include forbidden colors to the problem.

3.1.3. Graph Coloring and the CAMPUS 02 Timetabling Problem

As already mentioned the CAMPUS 02 timetabling problem can be formulated as a class-teacher timetabling problem with some special constraints. One major problem in using the bipartite graph formulation from Theorem 3.1 are the courses that are taught by several teachers simultaneously.

One way to model this is to consider bipartite hypergraphs, and try to find colorings of the hyperedges. We do not follow this approach in further detail. We try to apply the concept of a conflict graph similar to the one in Definition 3.4, to obtain a vertex coloring problem with connections to the CAMPUS 02 timetabling problem. In addition to this blocking of lectures must be enforced for most courses. This is why the interval coloring problem is defined (see Definition 3.5).

If the block sizes for each course would be a fixed partition of *CourseHours*, a conflict graph could be defined, with these blocks as nodes and a variation of an interval coloring of this graph would correspond directly to solutions to the problem. We performed some experiments using this connection by generating a fixed arbitrary partition of *CourseHours* in the beginning and then calculating solutions based on this partition. This approach is discarded, because of the bad quality of the produced timetables.

Having this variability in the number and length of the final blocks for each course it is not possible to define a static graph where interval colorings of the nodes directly correspond to planned blocks. Nevertheless we can still look at a *course conflict graph* with the courses as its nodes, that encodes conflicts between courses. The number of time slots in the meetings of the courses can be introduced in this graph as node weights. This graph encodes many of the conflicts between different courses in a structured way and is useful in defining heuristics based on graph coloring heuristics.

Definition 3.6 (event conflict graph)**.** Given the CAMPUS 02 timetabling problem the *course conflict graph* $G = (V, E)$ is defined with vertex set $V := C$ equivalent to the set of courses and for every vertex $c \in V$ the demand is set by $d(c) := \mathit{CourseHours}(c) + \mathit{ExamHours}(c)$ to the number of time slots and exam time slots contained in $m(c)$.

Because courses are taught by lecturers and they need to obey the no-clash constraints corresponding edges between each pair of vertices that correspond to courses taught by the same lecturer are added, that is

$$E_L := \{\{c_1, c_2\} \colon l \in L, l \in m(c_1), l \in m(c_2)\}.$$

Analogously edges between courses taught in the same class are needed to model the no-clash constraints for classes, that is

$$E_C := \{\{c_1, c_2\} \colon k \in K, k \in m(c_1), k \in m(c_2)\}.$$

The edge set is defined as $E := E_L \,\dot\cup\, E_K$ and degree functions for the subgraphs containing only the edges E_L or E_K are denoted by $\deg_L$ and $\deg_K$.

Solutions to the timetabling problem now correspond to the assignment of multiple intervals to each node of the graph. Many additional restrictions (for instance for handling exam blocks) need to be added to these interval assignments, for them to be equivalent to the CAMPUS 02 timetabling problem. Still the important fact is, that the edges of the event conflict graph correspond to conflicts between all the time slots in the meetings of the courses represented by the nodes. This motivates the use of these vertex degrees for course selection in constructive timetabling heuristics (see Section 4.1.1).

3.2. Complexity of the Timetabling Problem

In this section the computational complexity of the CAMPUS 02 timetabling problem is analyzed. As shown in Section 2.1 the problem is a generalization of a class-teacher timetabling problem. The class-teacher timetabling problem is among the first timetabling problems analyzed for its computational complexity. For the definitions of complexity classes and the terminology used we refer the reader to the standard book of Sipser [57]. A comprehensive list of NP-complete problems is contained in the standard book of Garey and Johnson [31].

With no additional constraints, Theorem 3.1 shows, that the class-teacher timetabling problem is equivalent to the edge coloring problem in a bipartite graph. For this special case, it is known from Theorem 3.2, that the number of time slots needed is equal to the maximum degree in the corresponding graph. A coloring using exactly Δ colors and based on that a timetable can be calculated in polynomial time, as it is shown by Theorem 3.5.

Theorem 3.5 (König [35])**.** *An edge coloring of a bipartite graph $G = (V_1 \,\dot\cup\, V_2, E)$ using $\Delta(G)$ colors can be calculated in polynomial time.*

Proof. The algorithm iteratively generates a coloring $c\colon E \to \{1, 2, \dots, \Delta\}$ of the edges. When coloring the edge $e = \{x, y\}$ with $x \in V_1, y \in V_2$ it determines the free colors $F(x)$ and $F(y)$ for the vertices, that is

$$F(v) := \{c \in \{1, 2, \dots, \Delta\}\colon c(e) \neq c \ \forall e \in \delta(v)\}.$$

$F(x), F(y) \neq \emptyset$ because $\delta(x), \delta(y) \leq \Delta$ and e has no color assigned.

If $F(x) \cap F(y) \neq \emptyset$ the edge e is colored with an arbitrary color in $F(x) \cap F(y)$.

Else choose colors $c_x \in F(x)$ and $c_y \in F(y)$. Let P be the maximal connected component containing x in the subgraph of G with edge set

$$\{e \in E\colon c(e) = c_x \text{ or } c(e) = c_y\}.$$

P is a path whose edges have alternating colors, starting in x with an edge of color c_y. Because this path starts in x, all vertices of P in V_2 are adjacent to an edge with color c_y, implying that y is not part of P. Because P is a maximum path with alternating colors, the coloring obtained by swapping c_x and c_y inside P is still feasible in G.

After this swap c_y is a free color in x and y and the edge e can be colored with c_y.

This procedure has polynomial running time, because it needs to color m edges and in each step the sets $F(x)$ and $F(y)$ can be calculated using $O(\Delta)$ steps. If necessary the path P and the color swap within can be performed in $O(n)$ time. This implies a worst case running time of $O(nm)$. □

Remark. Cole et al. [16] showed that there exists an algorithm with near-linear running time $O(m \log(\Delta))$ to calculate an edge coloring with Δ colors in a bipartite graph. This is currently the best running time known for this problem.

Even, Itai and Shamir [29] showed in 1975 that already a minor extension of the problem, which is very common in practical timetabling problems, leads to NP-hardness. They show that availability constraints lead to NP-completeness of the problem. The following theorem shows, that already a very simple restriction of the class-teacher timetabling problem, with availability constraints only for the lecturers, is NP-complete.

Theorem 3.6 (Even et al. [29])**.** *The basic class-teacher timetabling problem (see Section 1.3.1) with additional availability constraints for lecturers is NP-complete, even in the following restricted form.*

- *The set of times consists only of three times, that is $|T| = 3$.*
- *Classes are available at all times in T.*
- *Every lecturer is assigned to exactly one or no meeting with any class.*
- *There are only the following two types of lecturers:*

2-lecturers *are assigned to exactly two meetings with two different classes, where each of these meetings contains exactly one unassigned time slot. For these teachers there are availability constraints restricting them to only two of the three times in T.*

3-lecturers *are assigned to exactly three meetings with three different classes, where each of these meetings contains exactly one unassigned time slot. These teachers are available at all of the three times in T.*

The proof of the theorem is a technical reduction of 3SAT to this special timetabling problem. To show that Theorem 3.6 is sharp with respect to problem restrictions, Even et al. [29] prove, that the same problem, where all lecturers are 2-lecturers, is polynomially solvable using a simple branching procedure.

It is easy to see that the problem shown to be NP-complete in Theorem 3.6 is a special case of the CAMPUS 02 timetabling problem. The formal statement is given in the following corollary.

Corollary 3.1. *The problem deciding if a feasible solution to the CAMPUS 02 timetabling problem exists is NP-complete.*

Proof. The containment of the problem in the complexity class NP is obvious.

The NP-hardness of the problem follows by a reduction from the NP-complete problem of Theorem 3.6 to the CAMPUS 02 timetabling problem. In the following we show how to construct an instance of the CAMPUS 02 timetabling problem by specifying the input parameters explained in Section 2.1.1 based on an arbitrary instance of the problem from Theorem 3.6.

To keep the notation as simple and clear as possible the weekends and the constraints corresponding to weekends in the CAMPUS 02 timetabling problem are ignored in this reduction. This simplification can be achieved using the original problem formulation by creating an additional day for each existing day and adding these pairs to the set of weekends. If these new days are not added to the set of available days for classes DK the new problem is equivalent to a problem without weekends.

The set of days corresponds to the times and is identified with it, that is $D := T$. To enforce the fact that courses can only be planned on whole days and in this way the days correspond to time slots we set the lower and upper block sizes to the number of hours of a day, that is $l \equiv u \equiv |H|$. The set of classes K and lecturers L are also equal to the classes and lecturers of the given problem. Because there are no exams in the problem we set $\mathit{ExamHours} \equiv 0$. Because classes are available at all times $DK = D \times K$.

Based on that it is now necessary to create courses in a way that the 2-lecturers and 3-lecturers of the problem are encoded.

For each lecturer $l \in L$ that corresponds to a 2-lecturer, two courses c_1 and c_2 are added to C that correspond to the two meetings of the 2-lecturer with the two classes k_1 and k_2. Of course the lecturer is assigned to these courses, that is $(c_1, l) \in CL$ and $(c_2, l) \in CL$. Also the courses are added to the curriculum of the corresponding classes, that is $(c_1, k_1) \in CK$ and $(c_2, k_2) \in CK$. Because each of these courses has to be scheduled at exactly one day $\mathit{CourseHours}(c_1) = \mathit{CourseHours}(c_2) = |H|$. Because a 2-lecturer is unavailable at one time $t \in T$ the corresponding day is added to the set of NoGo-Days for the lecturer, that is $(t, l) \in \mathit{NOGO}$.

For each lecturer $l \in L$ that corresponds to a 3-lecturer, three courses c_1, c_2 and c_3 are added to C corresponding to the three meetings of the classes k_1, k_2 and k_3 with the lecturer l. Again $(c_1, l), (c_2, l), (c_3, l)$ are added to CL, $(c_1, k_1), (c_2, k_2), (c_3, k_3)$ are added to CK and $\mathit{CourseHours}(c_i) = |H|$ for $i = 1, 2, 3$.

Based on this construction it is obvious that there is a bijection between solutions to this instance of the CAMPUS 02 timetabling problem and solutions of the corresponding instance of the problem in Theorem 3.6. □

Remark. It is important to note that the reduction above shows only one reason for the NP-hardness of the CAMPUS 02 timetabling problem. This should be easy to observe because for the reduction for instance the blocking constraints of the problem are not used.

In addition we look at several further complexity results, that show the borderline between polynomial solvable special cases and NP-complete variations of educational timetabling problems. This gives some intuition about which kind of constraints lead to NP-hardness.

Another reason for hardness of timetabling problems is room assignment. As this is not part of the CAMPUS 02 timetabling problem we do not include these results. Carter and Tovey [15] analyze which cases of room assignment lead to NP-hardness and for which formulations solutions can be found in polynomial time.

Preassignment of time slots is also very common among timetabling problems, as it is also the case for the CAMPUS 02 timetabling problem. De Werra and Mahadev [24] analyze in detail for which kind of graphs the problem with preassignments still stays polynomially solvable.

One other aspect leading to NP-hardness, are lectures given to groups of classes. This is analyzed by de Werra et al. [23] and they are able to show that, if one lecturer gives lectures to at least three groups of classes, the class-teacher timetabling problem becomes NP-complete.

The blocking constraints present in the problem are another quite obvious source of NP-completeness. Ten Eikelder and Willemen [59] prove NP-completeness of the class teacher timetabling problem, if only blocks of length two are enforced for some classes. Because blocking is an essential part of the CAMPUS 02 timetabling problem and there are two kinds of interconnected blocking constraints (blocks for courses and a single block for each class on each day) blocking is one of the essential sources of the hardness of the problem. Theorem 3.7 illustrate that already blocks of length 2 lead to NP-hardness.

Theorem 3.7 (Ten Eikelder & Willemen [59])**.** *The class-teacher timetabling problem with only one class and teacher, but with blocking constraints for some pairs of time slots and restrictions on the times used for time slots, is NP-complete.*

When looking at more general types of school timetabling problems, like the one formulated for the ITC2011, Cooper and Kingston [17] analyze many different independent reasons for NP-completeness in these problems.

4. Heuristics

In this section heuristics to solve the CAMPUS 02 timetabling problem defined in Section 2.1 are developed. Section 1.4.1 contains a review of the current state of the art on educational timetabling heuristics. Methods that are shown to be efficient in literature are combined and modified to suit the problem.

Low-level heuristics which are used as building parts of all the heuristics in this thesis are developed in Section 4.1. These are classified into construction heuristics, that are used to generate the timetable and perturbation methods, which can be used in improvement heuristics like local search. Section 4.2 then combines the low-level heuristics into a heuristic for the CAMPUS 02 timetabling problem. In Section 4.3 the use of hyper-heuristics, based on the low-level heuristics from Section 4.1, to create solutions to the problem is explained. As already motivated in Section 1.4.1 the reason to develop hyper-heuristics is that these are more flexible and lead to good solutions for different kinds of inputs and also for modified problem versions.

Because the CAMPUS 02 timetabling problem has strong similarities to school timetabling problems, we mainly follow the approach of Pillay [49], who was among the first to apply modern hyper-heuristic techniques developed for the university course and the exam timetabling problem to school timetabling. In [49] generally applicable hyper-heuristics are developed and we show that a similar approach is applicable to the CAMPUS 02 timetabling problem in Section 4.3.2. As already mentioned in Section 2.1.3 the CAMPUS 02 timetabling problem is as multi-objective optimization problem. The hyper-heuristic approach is generalized to a multi-objective version in Section 4.3.3.

All the heuristics in this section work on a given instance of the CAMPUS 02 timetabling problem and can obtain information about feasibility of assignments to time slots. Section 4.4 explains how to store an instance of the problem and the current solution, and how to retrieve the informations needed by the heuristics based on these data structures efficiently. These data structures also handle the storage of solutions and partial solutions.

An important restriction during the timetable construction is, that only feasible blocks of time are assigned to the time slots. The data structures storing the instances allow the system to obtain feasible block lengths for courses. This is no easy problem, because if a block of certain length is planned, the number of unassigned time slots must still admit a feasible partition into blocks fulfilling the lower and upper bounds. A first approach with the partition of the time slots into fixed blocks at the beginning is discarded, because this leads to infeasible or bad timetables in many cases.

4.1. Low-Level Heuristics

This section contains the building parts, called low-level heuristics, of all the developed heuristics in this thesis. Heuristics for the problem consecutively use these low-level heuristics as sub-methods. They are classified into two general types, depending on the nature of their interaction with timetables. To clarify the terminology, we give definitions of these two types, based on Burke et al. [9].

Construction heuristic A method working on a partial timetable that assigns times to some unassigned time slot in a meeting of the problem. The result is again a timetable, which can still be a partial.

Perturbation heuristic A method changing a given timetable by modifying the time slot assignments in a way, such that the result is still a timetable. The slacks of completeness constraints stay the same or increase by the application of a perturbation heuristic.

The definition of perturbation heuristic is an extension to the one given in [9], where perturbation heuristics are only allowed to work on complete timetables. This extension is necessary, because we want to implement a hybrid heuristic which mixes construction and perturbation heuristics.

Remark. Not all publications use the same terminology, especially with respect to perturbation heuristics. For instance in the work of Pillay [49], the heuristic `Allocate`(a) is classified as a perturbation heuristic, although it plans an event at a feasible time slot in a partial timetable.

Note that partial timetables (see Definition 2.1) generated by construction heuristics always fulfill the no idle times constraints, that are part of the problem for all classes. This is not the case for many other constructive solvers in the literature, but allows us to enforce this constraint, while keeping the heuristics relatively easy to implement, without the need for additional data structures to enforce these constraints later on in construction. This allows us to solve larger problem instances, which is necessary for the CAMPUS 02 timetabling problem.

Also note that the KHE14 algorithm [34], which does not use this approach, is not able to calculate a solution to the CAMPUS 02 timetabling problem in our experiments, because the no idle time constraints for all classes lead to a too high memory consumption.

The following sections explain two types of construction heuristics and the perturbation heuristics used by the algorithms throughout this thesis.

4.1.1. Course Plan Construction Heuristics

The first construction heuristic approach, which is similar to most construction heuristics in the literature, starts by selecting a course using predefined rules and then assigning time slots of this course with feasible times. These heuristics are called *course plan heuristics* in this thesis.

Algorithm 4.1 is a general template for such a course plan heuristic for the problem. Every course plan heuristic starts by selecting a course c, for which there still exist unassigned time slots, calling the `selectCourse` method, as shown in line 1. Line 2 then checks if the selected course contains an exam and if this is the case and the corresponding exam time slots have no times assigned, the `planExam` method is called to assign times to the exam time slots in $m(c)$. This is already a heuristic decision to plan exams before the rest of the course. Else the block at line 3 executes the `planBlock` method, which should assign times to some of the unassigned time slots of $m(c)$. We call this method `planBlock`, because it is only allowed to assign a full feasible block of time to the meeting $m(c)$.

The reason for the decision to plan exams before the rest of the course is, that there are more and stronger constraints for the exam time slots compared to the other time slots of a course.

It is important to note that this is only a template for course plan heuristics. To obtain concrete low-level heuristics the behavior of the methods `selectCourse`, `planExam` and `planBlock` must be specified.

Algorithm 4.1: General template for the *course plan heuristics.*

```
1 $c$ := selectCourse()
2 if $c$ has an exam and it is not already planned then
      $success$ := planExam($c$)
3 else
      $success$ := planBlock($c$)
  return $success$
```

In the following different possible implementations are explained to obtain a pool of concrete construction heuristics.

We provide several different implementations of the `planBlock` method:

- Search all feasible blocks of time assignable to $m(c)$ at each day and assign one of these blocks chosen at random to the meeting $m(c)$.
- Assign the last feasible block of times to $m(c)$, where last means the block of time with the latest starting time.
- Assign the first feasible block of times to $m(c)$, where first means the block of time with the earliest starting time.
- A specialized version of choosing the last feasible block of times, where for each class only blocks during unused days are used, if it is not possible to find a block of times on a used day (see Algorithm 4.2).

At first all blocks of time that are feasible for assignment to the meeting $m(c)$ of the current course are determined, such that the day of that block is not empty in the timetable of class k (line 1). If this set of blocks is empty, another search is performed, including such empty days. Here the last 4 weekends are excluded in the search (line 2). Line 3 checks if a feasible block of times was found in line 1 or 2, and if not the method returns with an error. In line 4 a random block of times is assigned to $m(c)$, if a new day is started. Else in line 5 it is checked, if a block of times exists, that fills a day completely for the class k, and if so this block is assigned to $m(c)$. If no such block exists the last block found during the search is taken.

Algorithm 4.2: Implementation of the `planBlock` method of Algorithm 4.1 using already used days and weekends before starting new days as late as possible.

```
   Input: course $c$
   $k$ := class assigned to the course $c$
   $newday$ := false
1  $blockpool$ := feasible blocks for course $c$, such that there is already another block
   planned at the day of the block for class $k$, sorted with respect to their start and
   end.
2  if $blockpool = \emptyset$ then
       $blockpool$ := feasible blocks for course $c$ on Fridays and Saturdays excluding
       the last 4 weekends, sorted with respect to their start and end. $newday$ :=
       true
3  if $blockpool = \emptyset$ then
       return false
4  if $newday$ then
       Assign a random block out of $blockpool$ to the time slots of $m(c)$.
5  else
       if a block exists in blockpool that fills a day completely then
           Assign the last of these blocks to the time slots of $c$.
       else
           Assign the last block in $blockpool$ to the time slots of $c$.
   return true
```

If the construction heuristic should also be used to plan exams, Algorithm 4.3 gives an implementation of the `planExam` method. It assigns the last feasible block of times to the exam time slots of the selected course. An alternative is to leave this method unimplemented, if it is intended for this concrete heuristic to plan no exams. In this case it will only work, if the exam of the selected course is already planned.

The implementation of the `chooseCourse` method is the part, which is analyzed by most publications about constructive heuristics for educational timetabling. The reason is, that this method corresponds to node selection in the graph coloring problem (see

Algorithm 4.3: Implementation of the `planExam` method of Algorithm 4.1

```
Input: course $c$
$blockpool$ := All feasible blocks to plan the exam of course $c$.
if $blockpool = \emptyset$ then
    return $false$
Assign the last block in $blockpool$ to the exam time slots of $c$.
return $true$
.
```

Section 3.1). This is, why we adapt heuristics shown to be efficient for graph coloring, to implement this method. Most of the research about constructive heuristics for the university course and exam timetabling problem only analyzes this choice, as for instance shown by Azlan and Hussin [3]. The constructive heuristics used by Pillay [49] are also only differentiated by the implementation of the `chooseCourse` method and we use the same implementations based on graph coloring here.

Largest degree Choose a course with the largest number of time slots, for which the completeness constraint is not satisfied by the current partial timetable.

This can be interpreted as a maximum degree heuristic in a graph, if each time slot is seen as a vertex and connect the vertices corresponding to the same meeting.

Saturation degree Choose a course with the smallest saturation number among all courses, for which the completeness constraint is not satisfied by the current partial timetable. Here the saturation number is given by the quotient of the number of times, for which it is still feasible to assign them to a time slot of the course (ignoring idle times constraints) and the slack of the completeness constraint of the course.

Lecturer degree Among all courses for which the completeness constraint is not satisfied by the current partial timetable, choose one where a lecturer assigned to its resource slot has the maximum number of meetings.

This is equal to the degree $\delta_L(v)$ in the event conflict graph given by Definition 3.6.

Class degree Among all courses for which the completeness constraint is not satisfied by the current partial timetable choose one where the class assigned to its resource slot has the maximum number of meetings.

This is equal to the degree $\delta_K(v)$ in the event conflict graph given by Definition 3.6.

Random Choose a course for which the completeness constraint is not satisfied by the current partial timetable at random.

To enforce that all exams are planned in advance, all these selection rules are used only after the completeness constraints for all exam time slots are satisfied. Before that courses whose exams are not planned are chosen randomly. This again is an implementation of the heuristic to plan exams in the beginning.

The course selection rules for all the heuristics above, except for the saturation degree heuristic, are not directly influenced by the current partial timetable (except for the fact that completely planned courses are not selected any more). This means that these heuristics do not need any additional calculations during the execution. The saturation degree needs the calculation of the number of feasible time slots for each course, after each change of the timetable.

In addition to these heuristics based on graph coloring a special construction heuristic to plan exams is developed. It tries to place exams at the last possible free weekend, with a preference for Fridays, because exams on Fridays are considered better than exams on Saturdays according to the experts from CAMPUS 02 University of Applied Sciences. The details of the implementation are explained in Algorithm 4.4. It consists of a `selectCourse` method that randomly selects a course that still has unassigned exam time slots. Then the `planExam` method plans the exam for the course in the following way. The heuristic iterates over each weekend (fr, sa) in line 1. Then it checks if it is possible to plan the exam on *sa* (line 2) and *fr* (line 3), and if so adds these days to corresponding pools for selection. In line 4 the algorithm checks if a feasible Friday exists and if not assign the set of feasible Saturdays for selection. The heuristic then plans the exam at the last possible day in the selected set (the set of feasible Fridays if any exist; line 5).

Algorithm 4.4: Heuristic for planning exams at good positions as late as possible implementing Algorithm 4.1.

```
  Function selectCourse()
      return Random course c, where exam slots are still unassigned.
  Function planExam(c)
      $examp oolsa := \emptyset$
      $exampool := \emptyset$
1     foreach Friday fr do
          sa := Saturday next to fr.
2         if it is feasible to plan the exam of c at sa then
              examp oolsa.append(block on sa)
3         if it is feasible to plan the exam of c at fr then
              exampool.append(block on fr)
4     if $exampool = \emptyset$ then
          $exampool := exampoolsa$
5     Assign the first ExamHours(c) times of the last day in exampool to the exam
      time slots in $m(c)$.
  /* The planBlock method is not implemented here because it is never
     called in this type of heuristic.                                 */
```

4.1.2. Time Fill Construction Heuristics

The second generic kind of construction heuristic is given by Algorithm 4.5 and this kind of construction heuristic is called *time fill heuristic*. In line 1 it first selects a class and a consecutive block of times (k, b), for which the class is free, by calling the `selectBlock` method. Line 2 then calls the `planBlock` method for the pair (k, b). The implementations of this method try to assign (at least parts) of the block b to a meeting $m(c)$ of a course c: $(c, k) \in CK$.

This approach is not that common in publications applying hyper heuristics to educational timetabling. One reason might be, that when using this template, one cannot use similarities of the timetabling problem with graph coloring, in a straight forward manner, to derive heuristics. When developing the specialized heuristic for the CAMPUS 02 timetabling problem, we observe that this kind of heuristics is very valuable in the construction of good timetables. The approach is partially motivated by imitating heuristics currently applied by human experts, generating the timetables manually.

Pimmer and Raidl [51] published a concrete heuristic for high-school timetabling following a similar approach, with the difference that they try to fill a selected block of time for all classes simultaneously. They also note that this kind of approach is not very common in construction heuristics for educational timetabling. We know of no work that uses heuristics of this kind combined with course plan construction heuristics in a hyper-heuristic to solve educational timetabling problems.

We believe different time fill heuristics in addition to course plan construction heuristics, to be efficient for many practical timetabling problems. Algorithm 4.5 needs the implementation of the methods `selectBlock` and `planBlock` to obtain a concrete constructive heuristic. Additionally it contains a list of times called *badtimes*, which consist of blocks of time for which the heuristic could not find time slots for assignment before. This can be used in the concrete heuristic implementations to stop choosing blocks of time again, that already failed.

Algorithm 4.5: General template for construction heuristics choosing time slots first (time fill heuristic)

```
1 $(k, b)$ := selectBlock()
2 if not planBlock($k, b$) then
3     badtimes.append($(k, b)$)
      return false
  else
      return true
```

The time fill heuristics (Algorithm 4.5) are especially useful to plan courses at times in a way, that leads to good timetables with respect to different objectives. For instance one can fill up half full days or select specific days, like the days of the intensive week, for planning courses. We developed several construction heuristics based on the objectives and the structure of the CAMPUS 02 timetabling problem, using this template.

Fill intensive week A heuristic for filling up random free blocks of time during the intensive week of classes using the `planBlock` implementation of Algorithm 4.6.

Fill exam days Filling up days with exams planned in the first hours of that day using the `planBlock` implementation of Algorithm 4.6.

Fill small blocks Filling up small empty blocks on days in the class timetable. The details of this method are shown in Algorithm 4.7.

The `selectBlock` method determines for each class k all blocks of time, at which the class if free, that have length smaller or equal a given bound for small blocks. One of these pairs of class and block (k, b) is then selected at random.

In the `planBlock` method courses of the class k are searched, to which the block of time can be assigned (line 1). One of these courses is chosen at random and the block of time is assigned to its meeting (line 2).

A common problem in many time fill heuristics is, to decide how to fill the free blocks of time. If the blocks are big (for instance complete days) it is often not possible to fill them completely. One has to decide, when to look for a course, such that the block of time can be assigned completely to the course meeting, and when to only assign a subblock of the times to a course meeting and leave some times of the block unassigned. Algorithm 4.6 is a heuristic that decides based on the number of times (hours) in the block, if we should only fill it completely (for small blocks) or also consider partial assignments (for big blocks). If partial assignment is considered, this implementation ensures that the left over times are not too small blocks of time, for future assignments to be feasible for this rest. This is especially important in the practical test instances from CAMPUS 02 University of Applied Sciences, because there exist very few courses that allow assignment of blocks of times of length one or two.

To achieve this Algorithm 4.6 starts with the check, if the considered block is small (line 1). For small blocks line 2 determines all courses to which this block of times can be completely assigned. One of these courses is then randomly chosen and the block of times assigned to $m(c)$ (line 3). In the case of a large block (line 4) all possible courses and the number t of hours starting from the beginning of the block B, that can be assigned to these courses are determined (line 5). One of these pairs (c, t) is then chosen at random and the sub block of the first t times in B is assigned to $m(c)$ (lines 6 and 7).

Algorithm 4.6: Implementation of the `planBlock` method to fill selected blocks of time completely or partially according to a block size heuristic.

```
   Input: Block of times $b$, class $k$
   Initialization: $smallbound$ gives a bound on the lengths of blocks which have to
                   be completely filled by planBlock. Experiments showed that a
                   value of 3 gives a good performance for the test cases.
1  if $|b| \leq smallbound$ then
       /* Handle small blocks of time.                                    */
2      $coursepool := \{c \in C \colon (c,k) \in CK, b$ can be assigned to $c\}$
       if $coursepool = \emptyset$ then
           return $false$
3      Assign $b$ to the feasible empty time slots of a random course $c \in coursepool$.
       return $true$
4  else
       /* Handle big blocks of time.                                      */
5      $coursepool := \{(c,t) \in C \times \mathbb{N} \colon (c,k) \in CK, t$ consecutive times
                  from the beginning of $b$ can be assigned to time slots in $c\}$
       if $coursepool = \emptyset$ then
           return $false$
6      Choose $(c,t) \in coursepool$ at random.
7      Assign the first $t$ times of $b$ to the feasible empty time slots of $c$.
       return $true$
   .
```

Algorithm 4.7: Time fill heuristic to fill up small empty blocks of time in class timetables (implementation of Algorithm 4.5).

```
   Initialization: $smallbound$ gives a bound on the lengths of blocks which are
                   considered small and should be completely assigned to a single
                   course by this heuristic. Experiments showed that a value of 3
                   gives a good performance for the test cases.
   Function selectBlock()
       $smallblocks := \{(k,b) \in K \times 2^T \colon |b| \leq smallbound,$
                  $b$ is an empty block of times in the timetable of $k\}$
       return $random\ element\ (k,b) \in smallblocks$
   /* Get a class $k$ and a block of times $b$                             */
   Function planBlock(k, b)
1      $coursepool := \{c \in C \colon b$ can be assigned to time slots in $c\}$
       if $coursepool = \emptyset$ then
           return $false$
2      Assign $b$ to the feasible empty time slots of a random course $c \in coursepool$.
       return $true$
```

4.1.3. Perturbation Heuristics

There are several easy to implement perturbation heuristics common in literature, which can be applied to the problem. The main difference to most other timetabling problems, making their implementation more complicated for the CAMPUS 02 timetabling problem, are the very strict blocking constraints, which prohibit the change of times assigned to single time slots. We can only change the time slots corresponding to the assignment of a block of time or assign feasible blocks of time, to still obtain a feasible timetable. Additionally the no idle time constraints of each class are kept satisfied. When explaining the perturbation heuristics below, we will in each case point out how to achieve this.

For some of the perturbation heuristics the notion of the block graph for the current timetable and a fixed resource is useful. To define this graph it is helpful to look at the timetable given by the subset of meetings to which some fixed resource is assigned.

Definition 4.1. A *resource timetable* $\mathcal{T}_r$ for the resource r is defined as the subset of meetings $m(c)$, in which the resource is assigned to a resource slot, that is

$$\mathcal{T}_r := \{m(c) \colon r \in m(c)\}.$$

Based on the resource timetable the block graph for a resource can be defined. In addition to simplify the terminology a pair $(c, b) \in C \times 2^T$ is called a block of course c, if b is a block of times that is assigned to the meeting $m(c)$.

Definition 4.2. Given a resource r and its corresponding timetable $\mathcal{T}_r$, the digraph $G_r = (V, A)$ is called *block graph* of r, if the vertices are given by all blocks of time assigned to the meetings in $\mathcal{T}_r$,

$$V := \{(c, b) \colon b \text{ block of time in } m(c), r \in m(c)\}$$

and two such vertices are connected by an arc, if a block can be moved to the position of the other block,

$$\begin{aligned} A = \{((c_1, b_1), (c_2, b_2)) \colon & |b_1| = |b_2|, \\ & b_2 \text{ is a feasible assignment for the time slots assigned with } b_2 \text{ in } m(c_2)\}. \end{aligned}$$

The following is a list of common perturbation heuristics, adapted to the CAMPUS 02 timetabling problem:

Swap For a class $k \in K$ take two vertices $(c_1, b_1), (c_2, b_2)$ in G_k, that are connected by arcs in both directions in G_k, and swap the assigned blocks of time b_1, b_2 in the meetings $m(c_1)$ and $m(c_2)$.

Move For a class $k \in K$ take a block (c, b) planned as the last block on its day in $\mathcal{T}_k$, and assign to $m(c)$ a feasible block of time other than b.

Deallocate For a class $k \in K$ and a block (c, b) planned as the last block of the day in $\mathcal{T}_k$ and unassign the corresponding time slots in $m(c)$.

For all these simple perturbations there are different block selections to achieve different objectives. The following lists some of these possible concrete implementations and which objective is to improved by its application.

- **Swap**
  - Search for a block of times assigned to important courses, and consist of late times and check if it can be swapped with a block of another course, that is less important. Swap pairs of such blocks if they exist, else do nothing. This heuristic aims to improve the quality of the timetable with regard to didactic course placement.
  - Try to swap two courses in a way, such that a lecturer has to teach on less days, or the idle times of the lecturer are reduced.
- **Move**
  - Search free blocks of time in class timetables that should be filled and blocks at bad positions (for instance single blocks at days) and try to move these to the free blocks.
  - Check in lecturer timetables, if we can move some blocks to days at which the lecturer already teaches.
- **Deallocate**
  - Deallocate blocks that have many other positions valid for scheduling.
  - Deallocate blocks positioned at bad times for classes.
  - Deallocate blocks positioned at bad times for the lecturers.

Additionally to these simple perturbation heuristics we introduce a perturbation heuristic trying to move blocks using other same size blocks in between, which was motivated by some results using only construction heuristics. In these results there are sometimes days with only a single block (c, b) in the class timetables $\mathcal{T}_k$. Such blocks are called *bad blocks*. The heuristic tries to assign different feasible blocks of time b' instead of b to the corresponding time slots in $m(c)$, such that these newly assigned times where previously free times during a partially filled day in $\mathcal{T}_k$. This of course leads to a better timetable because the number of free days for the class increases and for another day more hours are assigned in the class timetable.

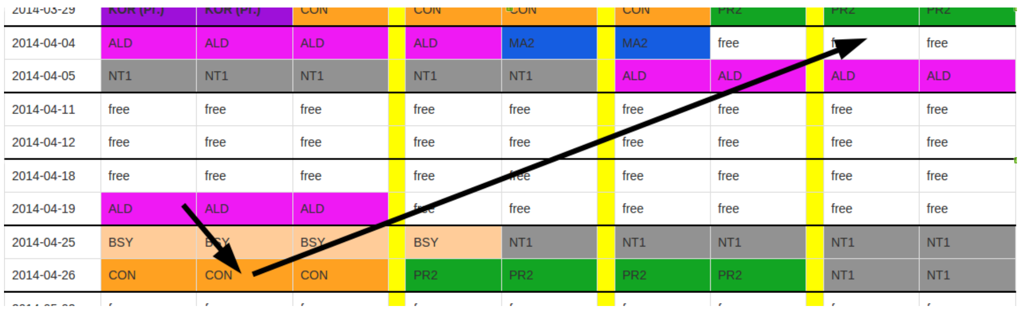
To achieve this, some class $k \in K$ a *bad block* (c, b) in $\mathcal{T}_k$ is selected and free blocks of time b' of same length are searched, that is $|b'| = |b|$, which are added as special nodes to the graph G_k, connecting other blocks to it if it is feasible to assign b' to their corresponding meeting. Now the algorithm searches for paths from the vertex corresponding to (c, b) to one of these new vertices. Moves of the blocks along this path are executed to obtain the desired result.

This method is explained in detail in Algorithm 4.8. At first (line 1) a block of times assigned to a course is selected, that should be moved. Other selection criteria are possible here, to optimize for different objectives. Then, starting with line 2, free blocks of time in $\mathcal{T}_k$ are determined, as possible new placements for a block of same length. These blocks of time are called good blocks. Here one can also add special selection criteria to optimize for different objectives. Starting with line 3 the good blocks are added to the block graph G_k. A path P from the selected block to a good block is searched in this graph using depth first (line 3). Then the blocks are are moved along this path (line 4).

Figure 4.1 illustrates the improvement method of this algorithm with a concrete example. The arrows show the path along which the blocks are moved.

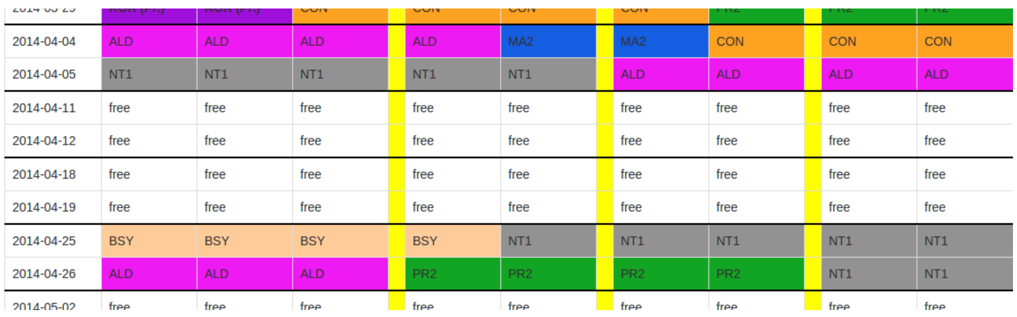
Algorithm 4.8: Perturbation heuristic moving blocks along paths in an extension of G_k.

```
   Input: Class $k \in K$
1  Select a block $(c, b)$ in $\mathcal{T}_k$ which is the only block on its day in $\mathcal{T}_k$ and
   $|b| < |H| - 1$.
   $goodblocks := \emptyset$
2  foreach day $d$ in $\mathcal{T}_k$ do
       if $\exists$ $|b|$ free time slots after the last block in $\mathcal{T}_k$ on $d$ then
           Add block of size $|b|$ corresponding to these times to $goodblocks$.
3  Add special vertices $(\emptyset, b)$ to $G_k$ corresponding to each block $b \in goodblocks$.
   Connect these vertices by adding the arcs
       $\{((c, b), (\emptyset, b')) \colon |b| = |b'|, b'$ is a feasible assignment for the time slots in $m(c)\}$.
4  Find a path $P$ from $(c, b)$ to a vertex $(\emptyset, b')$.
5  foreach $((c_1, b_1), (c_2, b_2)) \in P$ do
       Assign $b_2$ to the meeting $m(c_1)$.
```



2014-03-29	KOR (Pr.)	KOR (Pr.)	CON	CON	CON	CON	PR2	PR2	PR2
2014-04-04	ALD	ALD	ALD	ALD	MA2	MA2	free	free	free
2014-04-05	NT1	NT1	NT1	NT1	NT1	ALD	ALD	ALD	ALD
2014-04-11	free	free	free	free	free	free	free	free	free
2014-04-12	free	free	free	free	free	free	free	free	free
2014-04-18	free	free	free	free	free	free	free	free	free
2014-04-19	ALD	ALD	ALD	free	free	free	free	free	free
2014-04-25	BSY	BSY	BSY	BSY	NT1	NT1	NT1	NT1	NT1
2014-04-26	CON	CON	CON	PR2	PR2	PR2	PR2	NT1	NT1

(a) An extract of the timetable to which the heuristic is applied and the path along which the block is moved by the heuristic.



2014-03-29	KOR (Pr.)	KOR (Pr.)	CON	CON	CON	CON	PR2	PR2	PR2
2014-04-04	ALD	ALD	ALD	ALD	MA2	MA2	CON	CON	CON
2014-04-05	NT1	NT1	NT1	NT1	NT1	ALD	ALD	ALD	ALD
2014-04-11	free	free	free	free	free	free	free	free	free
2014-04-12	free	free	free	free	free	free	free	free	free
2014-04-18	free	free	free	free	free	free	free	free	free
2014-04-19	free	free	free	free	free	free	free	free	free
2014-04-25	BSY	BSY	BSY	BSY	NT1	NT1	NT1	NT1	NT1
2014-04-26	ALD	ALD	ALD	PR2	PR2	PR2	PR2	NT1	NT1
2014-05-02	free	free	free	free	free	free	free	free	free

(b) The result obtained after an application of the improvement heuristic.

Figure 4.1.: This is an example visualizing the perturbation heuristic moving blocks along paths.

4.2. A Heuristic for the CAMPUS 02 Timetabling Problem

In this section the low-level heuristics from Section 4.1 are combined to a heuristic for the CAMPUS 02 timetabling problem. This heuristic was developed in an iterative process, based on the real test data from the CAMPUS 02 University of Applied Sciences. It is a hybrid heuristic, as it uses construction heuristics from Section 4.1.1 and Section 4.1.2, and perturbation heuristics from Section 4.1.3. We call this heuristic the C02-heuristic.

Algorithm 4.9 is a selection order of the low-level heuristics from Section 4.1. The following enumeration explains how the low-level heuristics are selected, where the numbers refer to the corresponding line numbers in the algorithm. The enumeration also contains the motivation for these selections, based on the structure of the CAMPUS 02 timetabling problem.

1. Full assignment of the times during the intensive weeks to meetings of the corresponding class is a hard constraint of the problem. Satisfaction of this constraint cannot be guaranteed during construction of the timetable, as it is the case for the no idle times constraints or the no clash constraints.

   By using a time fill heuristic to assign these times to feasible meetings in the beginning, it is possible to fulfill these constraints. Such a time fill heuristic runs until the intensive week is filled for each class or no progress is made for multiple iterations.

2. Because all times assigned to time slots are constrained to be earlier than the times assigned to the exam time slots, it simplifies the construction, if exams are already fixed. Additionally exam time slots are also restricted to be assigned to the early times during each day, which makes it hard to plan them late in the construction process. This is the reason why exams are planned in advance in this second step, because else course time slots take away the feasible times for exam time slots, producing an infeasible timetable.

   This is achieved by calling the exam planning heuristic explained in Algorithm 4.4, until all exams are planned.

3. In a next step, days already containing exams in their class timetables $\mathcal{T}_k$ are filled, because full days are considered good in the objective function. Some of these days are after most other exams and therefore admit only a limited set of courses to assign their times. This is why we fill them before applying the general construction techniques.

   This is again achieved using a time fill heuristic, where the `selectBlock` method is implemented to choose only blocks on days, whose first hours are assigned to exam time slots.

4. This part is the main construction algorithm. It consists of two different low-level heuristics that are applied alternately.

   The main construction heuristic is an implementation of Algorithm 4.1 where **Saturation degree** is used for course selection and Algorithm 4.2 as the implementation of `planBlock`. This heuristic is executed 50 times, after which we switch to the second heuristic in this step.

   The second heuristic used in this construction step is a time fill heuristic, to fill small blocks of time completely on nearly full days in the class timetables $\mathcal{T}_k$. This is achieved by executing Algorithm 4.7 ten times, after which we switch back to the main construction heuristic above.

   This process is iterated until a complete timetable is found, or we observe too many iterations with no progress.

5. After looking at solutions produced by steps 1-5, we observed that some produced solutions had bad objective values because of single blocks on some days in $\mathcal{T}_k$. These blocks can often be moved away manually, by an approach similar to Algorithm 4.8. This motivated the development of this heuristic and the heuristic is applied to each complete timetable found by the steps 1-5.

Because in several low-level heuristics applied in Algorithm 4.9 choices are picked randomly, several runs of the heuristic are performed and the best solution, with respect to the objective function used, found among these executions is taken. It is known from graph coloring heuristics and other timetabling heuristics, that randomization can help to obtain better solutions. A similar behavior is observed for the CAMPUS 02 timetabling problem, as is discussed in more detail in Section 5.2.

Algorithm 4.9: C02-heuristic for the CAMPUS 02 timetabling problem.

```
1 while we can fill empty blocks in an intensive week do
      /* Fill the intensive week to fulfill the corresponding
         constraint.                                              */
      Apply the fill heuristic selecting an empty block of time during the intensive
      week and apply Algorithm 4.6.
2 while not all exam time slots are assigned do
      /* Plan all exams randomly as late as possible.             */
      Randomly plan exam using Algorithm 4.4.
3 while it is possible fill up days in $\mathcal{T}_k$ with exam blocks do
      /* Fill up days which already contain exam blocks.          */
      Apply the fill heuristic selecting an empty block on a day with an exam and
      apply Algorithm 4.6.
4 while all completeness constraints are not fulfilled do
      for $i := 1$ to 50 do
          /* Basic timetable construction.                        */
          Apply Algorithm 4.2 with Saturation degree course selection.
      for $i := 1$ to 10 do
          /* Fill up small blocks.                                */
          Apply Algorithm 4.7.
      if no progress then
          return false
5 repeat
      /* Move single blocks to incomplete days.                   */
      Apply Algorithm 4.8.
  until no change in last iteration;
  return true
```

4.3. A Hyper-Heuristic Approach

The literature review on heuristic methods for educational timetabling (see Section 1.4.1) suggests, that hyper-heuristic methods should be able to automatically generate good heuristics for arbitrary instances of the CAMPUS 02 timetabling problem, given a set of low-level heuristics. Pillay [49] applied several of the current state of the art hyper-heuristic techniques to school timetabling problems and achieved good results for several benchmark instances. This motivates the question, if a similar approach is able to produce competitive results for the CAMPUS 02 timetabling problem.

Hyper-heuristics are methods to generate new good heuristics for a problem. Burke et al. [9] define a hyper-heuristic as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems".

Because the hyper-heuristics developed in this thesis are based on genetic algorithms Section 4.3.1 contains an introduction to this topic. In Section 4.3.2 we combine the construction and perturbation selection hyper-heuristic of [49] into a hybrid hyper-heuristic, that allows the application of perturbation heuristics during the construction phase on partial timetables. This is a suggested extension of the hybrid heuristic in [49], which only applied perturbation heuristics after the construction of a complete timetable. The developed hyper-heuristic searches for a selection order of low-level heuristics using genetic algorithms.

As mentioned in Section 2.1 the timetabling problem is actually a multi-objective optimization problem. Different low-level heuristics (see Section 4.1) are implemented to improve different objectives of this problem. Section 4.3.3 extends the hyper-heuristic of Section 4.3.2 by using multi-objective optimization to obtain different selection strategies depending on the importance of the different objectives.

4.3.1. Introduction to Genetic Algorithms

The introduction to evolutionary algorithms given in this section is based on the books of Mitchell [44] and Bäck et al. [4].

Every genetic algorithm is composed of the following basic compounds.

Individual A representation of the objects in the search space of the optimization problem.

Fitness function A function that evaluates an individual and returns the corresponding objective function. The value obtained by this function is called fitness of the individual.

Initial population Description of the initialization of a first set of individuals.

***Selection* method** A method to choose the individuals of the population used as the next generation.

Genetic operators Descriptions of how to perform *mutation* and *crossover* of individuals.

Offspring After applying genetic operators to some subset of the population new individuals are generated in an iteration of the algorithm. These individuals are called offspring.

Algorithm 4.10 shows the generic form of a genetic algorithm. It starts by generating the initial population of individuals and evaluating their fitness using the fitness function. Then iteratively until some stopping criterion (e.g. a fixed number of iterations) is reached, the following steps are performed after each other:

1. For a subset of pairs of individuals the crossover operator is applied to generate a set of individuals called the offspring.
2. On another subset of individuals the mutation operator is applied to generate further individuals added to the offspring.
3. Now the fitness of all new individuals is evaluated.
4. A subset of individuals is selected to form the new generation (population for the next iteration).

Algorithm 4.10: Generic form of a genetic algorithm.

```
1 Generate the initial population.
2 Evaluate the fitness of the elements in the population
3 while A stopping criterion is not reached do
4     Perform crossover between the elements of the population.
5     Perform mutation on the elements of the population.
6     Evaluate the fitness of the new elements in the population.
7     Select the a subset of all new and old individuals for the next generation of
      the population.
```

A concrete version of a genetic algorithm is the $(\mu+\lambda)$ genetic algorithm. Because this version is used in the following sections we explain it in more detail (see Algorithm 4.11).

The algorithm starts as every genetic algorithm by initializing the initial population of μ individuals and evaluating the fitness function of its individuals (lines 1 and 2). It generates n generations of populations, where n is a fixed parameter (line 3). To construct the next generation based on the current population the offspring of the population is generated (line 4). The algorithm generates an offspring containing λ individuals, where with probability p_{cx} a new individual is generated by applying the crossover operation between two random individuals of the population and taking the first child (line 5). With probability p_{mut} the child is generated by applying the mutation operation to a random element of the population. In the other case (with probability $1-(p_{cx}+p_{mut})$)

the child for the offspring is generated by cloning a random element of the population (line 7). After offspring generation the fitness function of the individuals in the offspring is evaluated (line 8). Then the selection function is executed to select μ elements of the offspring and the current population as the next generation.

Algorithm 4.11: The $(\mu + \lambda)$ genetic algorithm.

```
1  pop := initial population.
2  Evaluate the fitness function of the individuals in pop.
3  for $i := 1$ to $n$ do
4      /* An offspring of $\lambda$ individuals is generated using the
          following method:                                          */
       offspring := $\emptyset$
       for $j := 1$ to $\lambda$ do
           $p$ := random($[0, 1]$)
           if $p \in [0, p_{cx})$ then
5              Add the first child of the crossover operator between two random
               individuals of pop to offspring.
           else if $p \in [p_{cx}, p_{mut})$ then
6              Add the result of the mutation of a random individual of pop to
               offspring.
           else
7              Add a random individual of pop to offspring.
8      Evaluate the fitness function of the individuals of offspring.
9      Apply the selection function to select $\mu$ elements of offspring $\cup$ pop for the
       next generation and assign it to pop.
```

A common selection method used in genetic algorithms is tournament selection, which is used in the heuristic presented in Section 4.3.2.

Tournament selection A fixed number (the tournament size) of individuals is selected from the population at random and among them the individual with the best fitness is selected.

4.3.2. A Hybrid Selection Hyper-Heuristic based on Genetic Algorithms

This section explains the implementation of a hybrid selection hyper-heuristic for school timetabling, extending the work of Pillay [49], using a genetic algorithm. The basics about genetic algorithms and their implementation are already explained in Section 4.3.1. The genetic algorithms in this thesis are implemented using the DEAP framework [30].

To define a genetic algorithm the individuals of the population are defined. An individual has to correspond to a selection of low-level heuristics. We encode such an individual using a string, where each symbol corresponds to a unique low-level heuristic from Section 4.1. These strings correspond to a full hybrid heuristic, if they are executed by running the heuristics corresponding to the symbols in the string after each other. After each step we check, if a complete timetable is obtained. Among all complete timetables found during the execution, the individual with the best objective value is chosen. The exams are planned in advance using the heuristic given by Algorithm 4.4. Algorithm 4.12 contains a detailed implementation of the individual execution.

Algorithm 4.12: Execution of an individual of the hybrid selection hyper-heuristic.

```
Initialization: best assigned to null value to store best result. Apply a fixed
                seed to the random number generator.
Input: individual string $s$
Execute the exam planning heuristic given by Algorithm 4.4, until all exams are
planned.
for $x \in s$ do
    Execute the low-level heuristic corresponding to $x$.
    if current timetable is complete then
        if objective value of current timetable is better than objective value of best
        then
            best := current timetable.
    return best and corresponding objective value
```

Because we mix construction and perturbation heuristics the choice for the length of the *individual* string is not obvious and is considered as a parameter. We took the number of low-level heuristic executions by the C02-heuristic (see Section 4.2) as a reference point and increased this value. A string length of 1500 leads to good results in the experiments for the practical test data. In addition it is possible to supply a special empty heuristic, which does not change the current timetable, to the hyper-heuristic, to allow the method itself to choose to use less iterations.

One important problem faced in this approach is, that many of the low-level heuristics use random decisions. This could lead to completely different fitness values for evaluations of the same individual. To get rid of these inconsistencies we set a fixed seed for the random number generator, before evaluation of the fitness function (Algorithm 4.12).

The *initial population* is generated, as recommended for most genetic algorithms, by generating random individuals. This is easy in this case, because it is only necessary to generate random strings of fixed length. The size of the initial population is a parameter of the algorithm.

As selection strategy tournament selection is used. Several different tournament sizes were tested as parameters of the genetic algorithm. Details of these results are listed in Section 5.2.3, which deals with the computational results obtained with the multi-objective hyper-heuristic.

As the *mutation* operator each entry of the string is changed with a probability p to a random low-level heuristic. The value $p = 0.01$ shows good results and is used for mutation. For the crossover operation a standard two point crossover of the two strings corresponding to the individuals is executed.

The $(\mu + \lambda)$ *evolution strategy* [5] is used (see Section 4.3.1, Algorithm 4.11 for details) to evolve the generations of selection heuristics. The number of generations produced is also an important parameter of the evolutionary algorithm.

Besides the genetic programming parameters, the major choice for a selection hyper-heuristic is the set of low-level heuristics given given to the algorithm. We tested the algorithm using two different subsets of low-level heuristics.

The first selection hyper-heuristic tested, selects only from the heuristics used by the C02-heuristic (Section 4.2) and a heuristic performing no change:

- Intensive week time fill heuristic.
- Specialized random exam heuristic (see Algorithm 4.4).
- Exam day fill heuristic.
- Construction heuristic based on **saturation degree** course selection and with Algorithm 4.2 for the `planBlock` method.
- Small block filling heuristic.
- Do nothing.

We refer to this set of heuristics with **H1**.

In a second test we allowed the selection hyper-heuristic to additionally choose from the following more general set of low-level heuristics:

- Basic construction heuristics with the `planBlock` method given by Algorithm 4.2 and course selection among:
  - **Largest degree**
  - **Saturation degree**
  - **Lecturer degree**
  - **Class degree**
  - **Random**
- Intensive week fill heuristic.
- Small block filling heuristic.
- Path improvement perturbation.
- Do nothing.

This set of heuristics is refered to by **H2**. It should allow evaluation of different course chosal methods, based on graph coloring.

4.3.3. Multi-Objective Optimization using Hyper-Heuristics and Genetic Algorithms

Because of the multi-objective nature of educational timetabling problems it is of special interest to apply multi-objective optimization to the problem. There are several genetic algorithms to tackle multi-objective optimization problems.

According to the literature review hyper-heuristic methods for multi-objective optimization in educational timetabling are not as well studied. Especially, we know of no work applying hyper-heuristics to develop heuristics for a multi-objective formulation of the school timetabling problem.

In the following we explain the motivation to apply a similar hybrid selection hyper-heuristic to a multi-objective formulation of the CAMPUS 02 timetabling problem.

From the results obtained with the selection hyper-heuristic from Section 4.3.2 (see Section 5.2) we conjecture, that the main advantage of applying genetic programming with hyper-heuristics is, that the genetic algorithm learns 'good random decisions' inside the low-level heuristics. We claim that a genetic algorithm can also do this for multiple objectives.

The aim when solving a multi-objective optimization problem is, to find Pareto-optimal solutions. We use the concepts as introduced by Ehrgott [28] and refer the reader to his book for further details about multi-objective optimization.

Definition 4.3. Let X denote the set of all feasible solutions and $f : X \to \mathbb{R}^n$ the multiple objective functions. A feasible point $x^* \in X$ is called *Pareto-optimal*, if there exists no other point $x \in X$ such that $f_i(x) \leq f_i(x^*)$ for all $i = 1, \dots n$ and $f_i(x) < f_i(x^*)$ for some $i \in \{1, \dots, n\}$.

If for two points $x, y \in X$ it holds that $f_i(y) \leq f_i(x)$ for all $i = 1, \dots n$ and $f_i(y) < f_i(x)$ for some $i \in \{1, \dots, n\}$ the point x dominates y.

To define a multi-objective hyper-heuristic a multi-objective fitness value has to be assigned to each individual. Here it is not possible to take the ''best" value among all feasible solutions generated in the execution as in Section 4.3.2. The most natural choice is to take the objectives of the last feasible solution generated during the execution of the individual, because this encodes the most information about the current individual.

In literature the NSGA-II [25] algorithm is known to perform well for multi-objective optimization problems. It uses a specialized multi-objective selection method to choose the individuals of the next generation. Additionally it also specifies a method to select individuals of the population for crossover and mutation, based on multi-objective principles, which is a generalization of tournament selection.

In the algorithm the individuals of the population are sorted into multiple fronts, with respect to Pareto-optimaltity. The individuals, which are dominated by no other individual of the population are assigned to front 1. Individuals which are only dominated by the individuals in front 1 are assigned to front 2. This process is contiued iterativly, and the index of the front, an individual is part of, is called the *rank* of the individual. The algorithm uses the rank of an individual in the current population as the major selection criterion. In addition to this rank individuals are also assigned a so called *crowding distance*, which is used to compare individuals on the same front. This is necessary, because in general there are many individuals on the same front. The crowding distance measures the closeness of an individual to its neighbors. We want this crowding distance to be large on average. The crowding distance is, in principle, the euclidean distance between the objective values of the two closest neighbors in the space of objective values, among the individuals with same rank. This way individuals on the border of a front are assigned infinite crowding distance.

For details on how to calculate the rank and crowding distance we refer the reader to the orginal publication of NSGA-II [25].

Based on these definitions the NSGA-II algorithm is explained, as shown in Algorithm 4.13. It starts by initializing the initial population (line 1). To generate the next generation of the population the Algorithm starts by selecting parts of the current population as parents for the offspring using binary tournament selection, based on the rank and crowding distance defined above (line 2), where greater crowding distance is chosen if the rank is the same. For those parents the crossover operation is performed with probability p_{cx}, and mutation is also performed (lines 3 and 4). The new individuals are then evaluated (line 5). In the method `selNSGA2` the population and offspring are again sorted into fronts to calculate the rank and the crowding distances, including the offspring. The new population is then selected based on these ranks and crowding distances.

Algorithm 4.13: Overview of the NSGA-II algorithm.

```
   Initialization: best assigned to null value to store best result. Apply a fixed
                   seed to the random number generator.
   Input: individual string $s$
1  pop := initialize population.
   for $j := 1$ to $n$ do
2      offspring := selectTournamentDCD(pop)
3      Build pairs of individuals in offspring and perform crossover with probability
       $p_{cx}$.
4      Perform mutation on the new individuals.
5      Evaluate new individuals in offspring.
6      pop := selectNSGA2(pop $\cup$ offspring)
```

4.4. Data Structures for Instances of the CAMPUS 02 Timetabling Problem

The basic idea of the data structure to store timetables is, that synchronized versions of the current timetable are stored for each resource (class or lecturer). The reason is, that most of the constraints (like the no clash constraints) are about resources and satisfaction of them can be answered when knowing the resource and the corresponding timetable efficiently.

Because the set of times T of the problem is partitioned into days $d \in D$, containing a constant small set of hours H, this partition into days is used to efficiently store the resource timetables. For each resource a dictionary indexed by days d is kept, in which we store all the blocks of time b, where all times in b are on the day d and the associated course c, such that $b \subseteq m(c)$. This allows for easy checks of no clash constraints, no idle time constraints and other blocking constraints.

Additionally we store for each course c the corresponding blocks of time assigned to $m(c)$. This information is kept synchronized.

This structure allows for efficient feasibility checks for most of the constraints. This is necessary, because in most of the heuristics from Section 4.1 sets containing all blocks of time, that are feasible for assignment to a given meeting, are generated.

5. Computational Results

This section summarizes the computational results obtained using the methods introduced in this thesis for practical problem instances of the CAMPUS 02 timetabling problem. To allow the comparison of the methods two representative practical data sets are chosen and the results obtained by applying the different methods to these instances are discussed and compared in the following. One of the chosen instances corresponds to a summer term and the other to a winter term. Section B contains the complete input for the 'winter term 2015/16' instance used.

To allow the comparison of different solutions the values of usedWeekends, unusedHours and freePeriods are shown for all classes. Because optimizing these values is essential for obtaining good timetables the sum of these objectives over all classes is an important objective function which is used to compare results of different methods and is shown in Equation 5.1.

$$\sum_{k \in K} (\text{usedWeekends}(k) + \text{unusedHours}(k) + \text{freePeriods}(k)) \tag{5.1}$$

Because this objective only incorporates wishes of classes Equation 5.2 sums the number of days at which all lecturers have to teach, to evaluate the quality of the solution from the perspective of lecturers. This objective is used in addition to Equation 5.1 for the multi-objective approach. It is also evaluated for all solutions discussed in this section.

$$\sum_{l \in L} \text{daysUsed}(l) \tag{5.2}$$

All experiments in this section were conducted using a Lenovo ThinkPad T440s with an Intel Core i7-4600 CPU and 12 GB of RAM.

5.1. Results using General Purpose Integer Programming Solvers

The integer programming model from Section 2.2 is implemented using AMPL (see Appendix A.2). Experiments using practical data sets like the one presented in Appendix B were performed using the general purpose integer programming solver IBM ILOG CPLEX version 12.6.1.0.

During the experiments it was not possible to solve any practical instance of the CAMPUS 02 timetabling problem to optimality. Feasible integer solutions are found by CPLEX after more than one hour, with the running time depending on the objective function used. Because of that only the sum of usedWeekends was minimized. These first feasible solutions are of no practical use, because of very bad objective values for all important objectives of the problem. Table 5.1 shows some of the objectives of such solution.

This motivates the study of heuristic methods to obtain good feasible solutions for the problem.

Input data	summer term 2014		
Solution method	CPLEX		
Running time	$\sim$ 90 min		
Objective (Eq.5.1)	253		
Lecturer Objective (Eq. 5.2)	457		
		Objectives	
Class	usedWeekends	unusedHours	freePeriods
2. term BSc	18	32	0
4. term BSc	17	45	1
6. term BSc	17	58	0
2. term MSc	17	48	0
Input data	winter term 2015/16		
Solution method	CPLEX		
Running time	$\sim$ 105 min		
Objective (Eq.5.1)	363		
Lecturer Objective (Eq. 5.2)	447		
		Objectives	
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	17	62	0
3. term BSc	17	74	0
5. term BSc	17	100	0
1. term MSc	17	59	0

Table 5.1.: Different objective values of the first integer solution found by CPLEX for the problem in Appendix B

5.2. Results obtained by the Heuristics

In this section we discuss the results of the heuristics presented in Section 4.

For all implemented heuristics the results obtained for two representative practical test cases ('summer term 2014' and 'winter term 2014/15') are analyzed.

5.2.1. Results from the C02-Heuristic

At first the results obtained with the C02-heuristic for the CAMPUS 02 timetabling problem are summarized. The heuristic often produces good results, depending on the random decisions made during the construction process. In the table below different objective values of runs of the heuristic are shown. For these results the heuristic was executed several times and the solution with the minimum value of the class objective function given in Equation 5.1 is chosen.

We choose this objective, because it is very important for timetables to be acceptable.

Table 5.2 and Table 5.3 show statistics of solutions obtained by this method with 10 and 100 runs.

Input data	summer term 2014
Heuristic	C02-heuristic (see Section 4.2)
# of runs	10
Running time	2 min 57 s
Objective (Eq.5.1)	91
Lecturer Objective (Eq. 5.2)	432

		Objectives	
Class	usedWeekends	unusedHours	freePeriods
2. term BSc	16	15	1
4. term BSc	14	6	5
6. term BSc	13	0	2
2. term MSc	12	6	1

Input data	winter term 2015/16 (see Appendix B)
Heuristic	C02-heuristic (see Section 4.2)
# of runs	10
Running time	2 min 37 s
Objective (Eq.5.1)	105
Lecturer Objective (Eq. 5.2)	406

		Objectives	
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	15	18	0
3. term BSc	16	13	0
5. term BSc	15	7	2
1. term MSc	16	3	0

Table 5.2.: Statistics for solutions of the C02-heuristic from Section 4.2 using 10 runs.

Input data	summer term 2014
Heuristic	C02-heuristic (see Section 4.2)
# of runs	100
Running time	42 min 56 s
Objective (Eq.5.1)	77
Lecturer Objective (Eq. 5.2)	428

	Objectives		
Class	usedWeekends	unusedHours	freePeriods
2. term BSc	15	6	1
4. term BSc	14	5	3
6. term BSc	12	0	2
2. term MSc	13	6	0

Input data	winter term 2015/16 (see Appendix B)
Heuristic	C02-heuristic (see Section 4.2)
# of runs	100
Running time	43 min 51 s
Objective (Eq.5.1)	96
Lecturer Objective (Eq. 5.2)	398

	Objectives		
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	16	10	0
3. term BSc	15	14	0
5. term BSc	13	5	0
1. term MSc	16	7	0

Table 5.3.: Statistics for solutions of the C02-heuristic from Section 4.2 using 100 runs.

It is observable that the timetables obtained after 100 runs of the heuristic are considerably better. This supports the claim from Section 4.2, that randomization is an efficient tool for tackling timetabling problems. It is important to note that many of the runs of the heuristic do not lead to feasible solutions. This would be a problem for purely deterministic heuristics, because some inputs would then lead to no solutions. Randomization is again the tool to get more stable behavior for different inputs. In Table 5.4 the number of runs leading to feasible timetables among the 100 executions, for the two test instances shown, are summarized.

Input data	# of runs leading to feasible solutions
summer term 2014	94
winter term 2015/16	40

Table 5.4.: Number of runs leading to feasible solutions among 100 runs of the C02-heuristic of Section 4.2.

The results indicate, that the constraints for the input data of 'summer term 2014' are easier to satisfy, than for the 'winter term 2015/16' test case. This is expected because for the 'winter term 2015/16' there are less times available for planning for each class and this leads to less feasible times to schedule the courses.

Another important aspect of the C02-heuristic from Section 4.2 is the application of the improvement heuristic (Algorithm 4.8) after the completion of a feasible timetable. Table 5.5 shows the average improvement of the objective (Equation 5.1) obtained, for all feasible solutions among the 100 runs of the C02-heuristic. These results support the claim that perturbation heuristics are important to obtain good timetables.

Input data	(a)	(b)	(c)
summer term 2014	6.76	88	77
winter term 2015/16	12.75	105	96

Table 5.5.: Improvement of the objective in Equation 5.1 from Algorithm 4.8 among 100 runs of the C02-heuristic. Column (a): Average improvement of the objective given by Equation 5.1 among all runs. Column (b): Minimum objective value (Equation 5.1) obtained among the 100 runs before applying the improvement heuristic. Column (c): Minimum objective value (Equation 5.1) obtained among the 100 runs after applying the improvement heuristic.

Comparing the running times and results of the C02-heuristic with the results obtained with integer programming (see Section 5.1), we support the general believe that heuristic methods are favourable to solve hard timetabling problems.

5.2.2. Results from the Selection Hyper-Heuristic

In this section we summerize the results obtained using the hybrid selection hyper-heuristic from Section 4.3.2. We can support the claim of Pillay [49], that hyper-heuristic methods perform well for school timetabling problems, for the case of the CAMPUS 02 timetabling problem. One still has to point out that it is necessary to implement data structures suitable to handle the special constraints of the problem. Also the low-level heuristics were modified to better suit the CAMPUS 02 timetabling problem.

At first the results obtained using the heuristic set **H1** are discussed. Experiments show that the following parameters lead to good performance of the algorithm:

- Individual string length of 1500.
- Number of generations: 10.
- Population size $\mu = 20$.
- $\lambda = 10$.
- $p_{cx} = 0.6$.
- $p_{mut} = 0.2$.
- A probability of 0.01 to mutate each character in the indiviual, for the mutation function.

Table 5.7 and 5.8 illustrate the progress in typical runs of the genetic algorithm for the input shown in Appendix B and the 'summer term 2014' test case. The solutions obtained using the best individuals are summarized in Table 5.9.

Table 5.6 shows a comparison between the results obtained with the C02-heuristic and the results of this hyper-heuristic method. This indicates that the genetic algorithm is a way to optimize good random decisions. This way good solutions can be calculated more quickly.

	summer term 2014		winter term 2014/15	
method	objective	running time	objective	running time
C02-heuristic (10 runs)	91	2 min 57 s	105	2 min 37 s
C02-heuristic (100 runs)	77	42 min 56 s	96	43 min 51 s
Selection hyper-heuristic (**H1**)	78	12 min 52 s	100	13 min 11 s

Table 5.6.: Comparison of the results obtained with the C02-heuristic and the selection hyper-heuristic using heuristic set **H1**. The objectives are calculated using Equation 5.1.

Generation	avg. fitness	std. deviation	min. fitness	max. fitness
0	112.4	5.24	106	119
1	105.6	2.80	100	107
2	104.2	3.43	100	107
3	106.0	8.92	100	123
4	101.4	2.80	100	107
5	100.0	0.00	100	100
6	100.0	0.00	100	100
7	100.0	0.00	100	100
8	100.0	0.00	100	100
9	100.0	0.00	100	100
10	100.0	0.00	100	100

Table 5.7.: Progress during genetic algorithm for hyper-heuristic from Section 4.3.2 with heuristic set **H1** for the input from Appendix B. The individuals have a length of 1500.

Generation	avg. fitness	std. deviation	min. fitness	max. fitness
0	200079.0	399960.50	93	infeasible
1	95.8	5.27	88	104
2	88.2	1.17	86	89
3	86.2	4.26	78	89
4	84.8	4.26	78	89
5	82.4	5.39	78	89
6	78.0	0.00	78	78
7	78.0	0.00	78	78
8	78.0	0.00	78	78
9	78.0	0.00	78	78
10	78.0	0.00	78	78

Table 5.8.: Progress during genetic algorithm for hyper-heuristic from Section 4.3.2 with heuristic set **H1** for the input 'summer term 2014'. The individuals have a length of 1500.

Input data	summer term 2014
Heuristic	Best individual from selection hyper-heuristic using heuristic set **H1**.
Running time	12 min 52 s
Objective (Eq.5.1)	78
Lecturer Objective (Eq. 5.2)	412

	Objectives		
Class	usedWeekends	unusedHours	freePeriods
2. term BSc	16	10	1
4. term BSc	14	3	2
6. term BSc	13	0	2
2. term MSc	12	4	1

Input data	winter term 2015/16 (see Appendix B)
Heuristic	Best individual from selection hyper-heuristic using heuristic set **H1**.
Running time	13 min 11 s
Objective (Eq.5.1)	100
Lecturer Objective (Eq. 5.2)	408

	Objectives		
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	16	19	0
3. term BSc	16	14	0
5. term BSc	14	0	0
1. term MSc	15	6	0

Table 5.9.: Statistics for solutions obtained using the best individual from a hyper-heuristic using heuristic set **H1**.

It is of interest to look at the distribution of the low-level heuristics in these best individuals. Table 5.10 shows the distribution for the two individuals, that lead to the solutions shown in Table 5.9. We observe that the low-level heuristics are relatively equally distributed. This suggests the hypothesis, that the genetic algorithm is mainly minimizing with respect to 'good random choices' inside the low-level heuristics, because the seed is fixed for each fitness evaluation. To validate this hypothesis we performed 100 runs of these best individuals. We observe that the selection of heuristics is good in a sense, that most of the evaluations lead to feasible solutons. This is not the case for the C02-heuristic from Section 4.2. The average and minimum objective value are shown in Table 5.11. But overall the hypothesis can be confirmed, because most of the results do not come close to the excellent objective value obtained by the fixed random choices, found by the genetic algorithm.

Input data	H1	H2	H3	H4	H5	H6
summer term 2014	17.5%	17.0%	16.1%	16.1%	16.5%	16.7%
winter term 2014/15	15,7%	17,1%	17,4%	18,3%	14,9%	16,6%

Table 5.10.: Distribution of the heuristics from the set **H1** in the individuals producing the results from Table 5.9.

Input data	avg. fitness	min. fitness	# feasible
summer term 2014	96.7	81	100
winter term 2014/15	113.0	92	87

Table 5.11.: Average and minimum fitness among 100 runs of the heuristics from the set **H1** in the individuals producing the results from Table 5.11.

Using the selection hyper-heuristic with heuristic set **H2**, similar behavior can be observed.

Tables 5.12 and 5.13 show how the heuristic set **H2** performs for the same test cases as used for the set **H1** above. Comparing with Tables 5.7 and 5.8 the similar behavior is obvious. But it is notable that the convergence is slower, because of the larger set of heuristics.

This behavior suggests, that when applying hyper-heuristics one should remove obviously inferior heuristics manually. If it is not possible to do these decisions in advance it is recommended to try running the algorithm with different subsets of the low-level heuristics, because working with too many heuristics slows down the genetic programming algorithm.

Generation	avg. fitness	std. deviation	min. fitness	max. fitness
0	550055.3	497432.64	114	infeasbile
1	150106.0	357026.89	114	infeasbile
2	118.3	6.71	114	137
3	113.7	2.17	109	117
4	111.3	2.49	109	114
5	110.0	2.00	109	114
6	109.8	1.79	109	114
7	109.0	0.00	109	109
8	109.0	0.00	109	109
9	109.0	0.00	109	109
10	109.0	0.00	109	109

Table 5.12.: Progress during genetic algorithm for hyper-heuristic from Section 4.3.2 with heuristic set **H2** for the input from Appendix B. The individuals have a length of 1500.

Generation	avg. fitness	std. deviation	min. fitness	max. fitness
0	50092.7	217923.69	82	infeasible
1	90.5	7.45	82	105
2	86.9	5.77	82	95
3	82.5	5.18	72	95
4	81.0	5.18	72	91
5	81.0	3.00	72	82
6	80.5	3.57	72	82
7	79.5	4.33	72	82
8	77.0	5.00	72	82
9	74.5	4.33	72	82
10	72.0	0.00	72	72

Table 5.13.: Progress during genetic algorithm for hyper-heuristic from Section 4.3.2 with heuristic set **H2** for the input 'summer term 2014'. The individuals have a length of 1500.

Input data	summer term 2014
Heuristic	Best individual from selection hyper-heuristic using heuristic set **H2**.
Running time	10 min 50 s
Objective (Eq.5.1)	72
Lecturer Objective (Eq. 5.2)	394

		Objectives	
Class	usedWeekends	unusedHours	freePeriods
2. term BSc	16	8	0
4. term BSc	15	0	1
6. term BSc	12	0	4
2. term MSc	13	2	1

Input data	winter term 2015/16 (see Appendix B)
Heuristic	Best individual from selection hyper-heuristic using heuristic set **H2**.
Running time	15 min 11 s
Objective (Eq.5.1)	109
Lecturer Objective (Eq. 5.2)	384

		Objectives	
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	16	19	0
3. term BSc	15	19	1
5. term BSc	15	0	3
1. term MSc	17	4	0

Table 5.14.: Statistics for solutions obtained using the best individual from a hyper-heuristic using heuristic set **H2**.

Input data	H1	H2	H3	H4	H5	H6	H7	H8	H9
summer term 2014	10.9%	11.1%	12.3%	11.3%	10.1%	10.6%	11.5%	9.7%	12.5%
winter term 2014/15	11.1%	11.3%	10.5%	10.8%	10.9%	10.8%	11.5%	12.1%	10.9%

Table 5.15.: Distribution of the heuristics from the set **H2** in the individuals producing the results from Table 5.14.

5.2.3. Results from the Multi-Objective Hyper-Heuristic

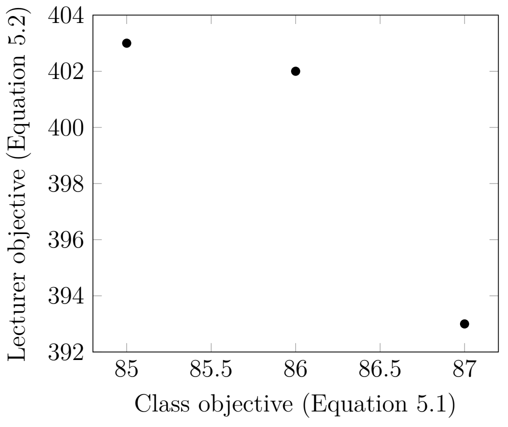
Because the problem is already very hard in the single objective case, only one additional objective is added to the problem. We follow the approach of Burke et al. [7], using weighted sums of objectives of different parties. For the classes we again use the objective given by Equation 5.1. Another objective, minimizing the number of days lecturers have to teach is added to the problem. This additonal objective is given by Equation 5.2.

Because the results for single objective hyper-heuristics suggest, that the main factor of improvement in the genetic algorithm is not the selection of the low-level heuristics, but the optimization of the random choices inside these heuristics, we again use the heuristic set **H1** for the multi-objective hyper-heuristic.

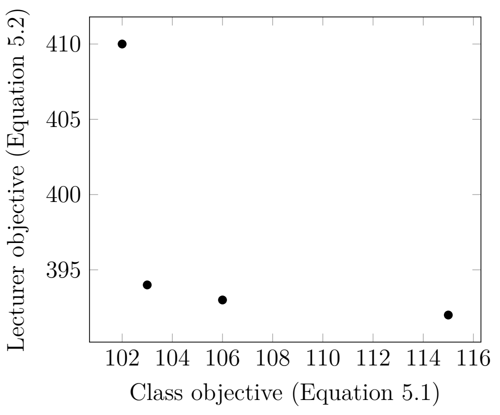
Figure 5.1 visualizes the non-dominated solutions obtained in a run of the hyper-heuristic. The pictures suggest the common form of a Pareto-front but it is obvious that the calculations were not sufficient to really reconstruct the full set of Pareto-optimal solutions. For these evaluations the following parameters were used, which showed to be efficient among several experiments.

- Number of generations: 10
- Population size: 12
- $p_{cx} = 0.6$

To obtain more details of the Pareto-optimal solutions calculations with larger populations and multiple initial populations could be performed. The long running times suggest, that the CAMPUS 02 timetabling problem is too hard to allow the complete calculation of the Pareto-front using these heuristic techniques.



(a) Input data: 'summer term 2014'



(b) Input data: 'winter term 2015/16'

Figure 5.1.: Non-dominated objective values from runs of the multi-objective hyper-heuristic.

This type of algorithm is still useful, because it allows a system to display multiple solutions, balancing the different objectives, to the user.

Ideally one could optimize with respect to each stakeholder, but this leads to too many objectives, which cannot be handled by current state of the art genetic algorithms. Further research is needed to confirm the performance of multi-objective hyper-heuristics for other educational timetabling problems.

6. Summary and further Research Directions

The results obtained using the heuristics of Section 4 and analyzed in Section 5.2 confirm several claims about timetabling problems from the literature.

The C02-heuristic from Section 4.2 allows the fast computation of good timetables. The development of such specialized heuristics is known to perform well for practical timetabling problems in the literature.

Using a section hyper-heuristic based on genetic programming, we confirm that these methods are able to generate competitive heuristics. This can also be confirmed for a hybrid approach, which is a generalization of the methods shown by Pillay [49]. Based on the distribution of the selected basic heuristics we conjecture that the important factor optimized in the genetic algorithm are good random choices in the basic heuristics. This conjecture should be analyzed in more detail using different variants of timetabling problems.

For the multi-objective generalization of the selection hyper-heuristic we are able to obtain several solutions of similar quality with better performances for different objectives. But it is not possible to really obtain a good representation of the Pareto-front because of the size of the problem.

Based on these observations it would be of interest to extend the methods used in this theses to the general school timetabling problem introduced for the ITC2011. This would allow to evaluate the methods using many different benchmark instances. Many of these instances are also of smaller size, which would allow further evaluation of the multi-objective hyper-heuristic.

As already mentioned by Pillay [49], there are no publications known, that analyze constructive perturbation hyper-heuristics for educational timetabling problems. As these methods are known to show good results for other combinatorial optimization problems, this should also be evaluated.

Research results about many-objective optimization heuristics would be of special interest for educational timetabling problems, because of the natural existence of many objectives in these problems.

Bibliography

[1] L. N. Ahmed, E. Özcan, and A. Kheiri. Solving high school timetabling problems worldwide using selection hyper-heuristics. *Expert Systems with Applications* 42(13):5463 – 5471, 2015.

[2] H. Asmuni, E. K. Burke, J. M. Garibaldi, B. McCollum, and A. J. Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers & Operations Research* 36(4):981 – 1001, 2009.

[3] A. Azlan and N. Hussin. Implementing graph coloring heuristic in construction phase of curriculum-based course timetabling problem. In *Computers & Informatics (ISCI), 2013 IEEE Symposium on*, pp. 25–29. IEEE, 2013.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*, volume 1. CRC Press, 2000.

[5] H.-G. Beyer and H.-P. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing* 1(1):3–52, 2002.

[6] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research* 194(1):59–70, 2012.

[7] E. Burke, Y. Bykov, and S. Petrovic. A multicriteria approach to examination timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pp. 118–131. Springer Berlin Heidelberg, 2001.

[8] E. Burke, M. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *Evolutionary Computation, IEEE Transactions on* 16(3):406–417, June 2012.

[9] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pp. 449–468. Springer US, 2010.

[10] E. Burke, K. Jackson, J. H. Kingston, and R. Weare. Automated university timetabling: The state of the art. *The Computer Journal* 40(9):565–571, 1997.

[11] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics. *J Oper Res Soc* 64(12):1695–1724, Dec 2013.

[12] E. K. Burke, B. McCollum, P. McMullan, and A. J. Parkes. Multi-objective aspects of the examination timetabling competition track. In *Proceedings of PATAT*, pp. 3119–3126. Citeseer, 2008.

[13] V. Cacchiani, A. Caprara, R. Roberti, and P. Toth. A new lower bound for curriculum-based course timetabling. *Computers & Operations Research* 40(10):2466 – 2477, 2013.

[14] M. W. Carter. Or practice—a survey of practical applications of examination timetabling algorithms. *Operations Research* 34(2):193–202, 1986.

[15] M. W. Carter and C. A. Tovey. When is the classroom assignment problem hard? *Operations Research* 40(1-supplement-1):S28–S39, 1992.

[16] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in o(e logd) time. *Combinatorica* 21(1):5–12, 2001.

[17] T. Cooper and J. Kingston. The complexity of timetable construction problems. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pp. 281–295. Springer Berlin Heidelberg, 1996.

[18] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pp. 176–190. Springer Berlin Heidelberg, 2001.

[19] J. Csima. *Investigations on a time-table problem.* PhD thesis, 1965.

[20] D. Datta, K. Deb, and C. Fonseca. Multi-objective evolutionary algorithm for university class timetabling problem. In K. Dahal, K. Tan, and P. Cowling, editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pp. 197–236. Springer Berlin Heidelberg, 2007.

[21] D. de Werra. An introduction to timetabling. *European Journal of Operational Research* 19(2):151–162, 1985.

[22] D. de Werra. Extensions of coloring models for scheduling purposes. *European Journal of Operational Research* 92(3):474 – 492, 1996.

[23] D. de Werra, A. S. Asratian, and S. Durand. Complexity of some special types of timetabling problems. *Journal of Scheduling* 5(2):171–183, 2002.

[24] D. de Werra and N. Mahadev. Preassignment requirements in chromatic scheduling. *Discrete Applied Mathematics* 76(1–3):93 – 101, 1997. Second International Colloquium on Graphs and Optimization.

[25] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pp. 849–858. Springer Berlin Heidelberg, 2000.

[26] R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[27] J. K. Edmund Burke, Dominique de Werra. Applications to timetabling. In *Handbook of Graph Theory, Second Edition, Section 5.5*, pp. 530–562. Chapman Hall/CRC Press, 2013.

[28] M. Ehrgott. *Multicriteria optimization*. Springer Science & Business Media, 2006.

[29] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multicommodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pp. 184–193, Oct 1975.

[30] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13:2171–2175, jul 2012.

[31] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[32] D. M. Gay and B. Kernighan. Ampl: A modeling language for mathematical programming. *Duxbury Press/Brooks/Cole* 2, 2002.

[33] C. C. Gotlieb. The construction of class-teacher timetables. In *Proceedings of the IFIP Congress*, pp. 73–77, 1962.

[34] J. H. Kingston. Khe14: An algorithm for high school timetabling. In *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, E. Ozcan, EK Burke, B. McCollum (Eds.)*, pp. 498–501, 2014.

[35] D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen* 77(4):453–465, 1916.

[36] S. Kristiansen and T. R. Stidsen. A comprehensive study of educational timetabling-a survey. Technical report, Department of Management Engineering, Technical University of Denmark, 2013.

[37] M. Kubale. Interval vertex-coloring of a graph with forbidden colors. In D. de Werra and A. Hertz, editors, *Graph Colouring and Variations*, volume 39 of *Annals of Discrete Mathematics*, pp. 125 – 136. Elsevier, 1989.

[38] R. Lewis, B. Paechter, and B. McCollumn. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical report, Second International Timetabling Competition, 2007.

[39] M. Maashi, G. Kendall, and E. Özcan. Choice function based hyper-heuristics for multi-objective optimization. *Applied Soft Computing* 28(0):312 – 326, 2015.

[40] M. Maashi, E. Özcan, and G. Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications* 41(9):4475 – 4493, 2014.

[41] R. H. McClure and C. E. Wells. A mathematical programming model for faculty course assignments. *Decision Sciences* 15(3):409–420, 1984.

[42] B. McCollum, P. McMullan, E. K. Burke, A. J. Parkes, and R. Qu. The second international timetabling competition: Examination timetabling track. Technical report, Second International Timetabling Competition, 2007.

[43] G. L. Michael W. Carter. Recent developments in practical course timetabling. In E. Burke and M. Carter, editors, *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pp. 3–19. Springer Berlin Heidelberg, 1998.

[44] M. Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

[45] G. Neufeld and J. Tartar. Generalized graph colorations. *SIAM Journal on Applied Mathematics* 29(1):91–98, 1975.

[46] K. Papoutsis, C. Valouxis, and E. Housos. A column generation approach for the timetabling problem of greek high schools. *The Journal of the Operational Research Society* 54(3):pp. 230–238, 2003.

[47] N. Pillay. An overview of school timetabling research. *Proceedings of the international conference on the theory and practice of automated timetabling* p 321, 2010.

[48] N. Pillay. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *J Oper Res Soc* 63(1):47–58, Jan 2012.

[49] N. Pillay. A comparative study of genetic programming and grammatical evolution for evolving data structures. In *proceedings of the 2014 PRASA, RobMech, and AfLaT International Joint Symposium, 27-28 November 2014, Cape Town, South Africa*, pp. 115–121, 2014.

[50] N. Pillay. A survey of school timetabling research. *Annals of Operations Research* 218(1):261–293, 2014.

[51] M. Pimmer and G. Raidl. A timeslot-filling heuristic approach to construct high-school timetables. In L. Di Gaspero, A. Schaerf, and T. Stützle, editors, *Advances in Metaheuristics*, volume 53 of *Operations Research/Computer Science Interfaces Series*, pp. 143–157. Springer New York, 2013.

[52] G. Post, L. Di Gaspero, J. H. Kingston, B. McCollum, and A. Schaerf. The third international timetabling competition. *Annals of Operations Research* pp. 1–7, 2013.

[53] G. Post, J. Kingston, S. Ahmadi, S. Daskalaki, C. Gogos, J. Kyngas, C. Nurmi, N. Musliu, N. Pillay, H. Santos, and A. Schaerf. Xhstt: an xml archive for high school timetabling problems in different countries. *Annals of Operations Research* 218(1):295–301, 2014.

[54] R. Qu, E. Burke, B. McCollum, L. Merlot, and S. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling* 12(1):55–89, 2009.

[55] H. Santos, E. Uchoa, L. Ochi, and N. Maculan. *Annals of Operations Research* 194(1):399–412, 2012.

[56] J. Silva, E. Burke, and S. Petrovic. An introduction to multiobjective metaheuristics for scheduling and timetabling. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*, pp. 91–129. Springer Berlin Heidelberg, 2004.

[57] M. Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 2012.

[58] M. Sørensen and F. H. Dahms. A two-stage decomposition of high school timetabling applied to cases in denmark. *Computers & Operations Research* 43(0):36 – 49, 2014.

[59] H. ten Eikelder and R. Willemen. Some complexity aspects of secondary school timetabling problems. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pp. 18–27. Springer Berlin Heidelberg, 2001.

[60] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal* 10(1):85–86, 1967.

[61] E. Zitzler. *Evolutionary algorithms for multiobjective optimization.* PhD thesis, Diss. Technische Wissenschaften ETH Zürich, Nr. 13398, 1999, 1999.

[62] M. Čangalović and J. A. Schreuder. Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operational Research* 51(2):248 – 258, 1991.

A. Software

As part of the thesis a software system is developed to automatically solve the timetabling problem of the Campus 02 University of Applied Sciences (see Section 2.1). To allow a structured input of the problem instances and storage of the corresponding solutions, a database is used. The details of the database architecture are explained in Appendix A.1. For experiments with general purpose solvers an AMPL [32] model of the integer programming formulation in Section 2.2 is used. This model is given in Appendix A.2. The details about the implementation of the heuristics (see Section 4) using the Python programming language are explained in Appendix A.3.

Additionally a small GUI interface accessing the database is developed using PHP to allow data input and visualization of the solutions.

A.1. Database

Figure A.1 shows the ER-model of the database used to store instances of the CAMPUS 02 timetabling problem and solutions generated by the algorithms.

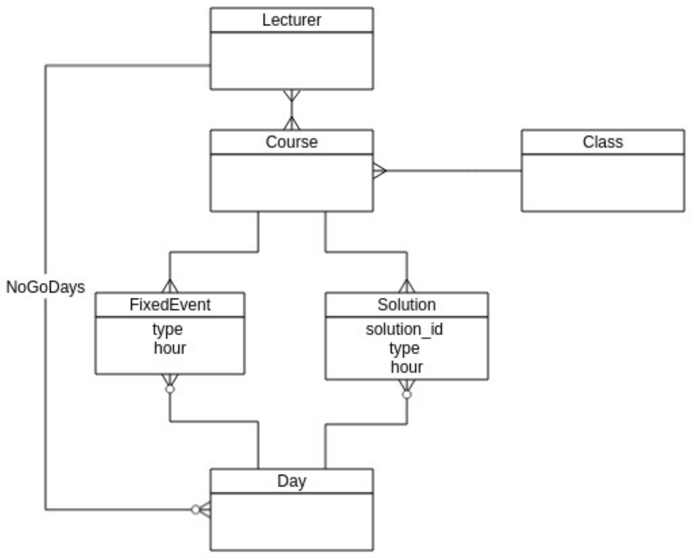


Figure A.1.: ER-model of the database.

A.2. Exact Modeling of the Integer Program using AMPL

```
set D ordered; # days
set H ordered default {1,2,3,4,5,6,7,8,9}; # hours per day
set L; # lecturers
set C; # courses
set K; # classes
set CK within {C, K};
set CL within {C, L};
set DK within {D, K};

set WEEKENDS within {d1 in D, d2 in D: ord(d2) = ord(d1)+1};
set WEEK within DK;

param CourseHours{C};
param LowBlock{C};
param UpBlock{C};
param ExamHours{C};

param Cost{D,H,C} default 1;

var x{D,H,C}, binary; # course times
var s{D,H,C}, binary; # course start times
var e{D,H,C}, binary; # exam times
var es{D,H,C}, binary; # exam start times

var day_s{K, D, H}, binary; # start of blocks each day for each class

var usedWeekend{WEEKENDS, K} binary;

minimize UsedWeekends:
    sum{(sa,so) in WEEKENDS, k in K} usedWeekend[sa,so, k];

subject to Lecturer{l in L, d in D, h in H}:
    sum{(c,l) in CL} (x[d,h,c] + e[d,h,c]) <= 1;

subject to Class{k in K, d in D, h in H}:
    sum{(c,k) in CK} (x[d,h,c] + e[d,h,c]) <= 1;

subject to CourseComplete{c in C}:
    sum{d in D, h in H} x[d,h,c] = CourseHours[c];

subject to ExamComplete{c in C}:
    sum{d in D, h in H} e[d,h,c] = ExamHours[c];

# if we have block on day it has to be inside bounds
subject to BlockLenLow{c in C, d in D}:
    sum{h in H} s[d,h,c] * LowBlock[c] <= sum{h in H} x[d,h,c];
# if we have block on day it has to be inside bounds
subject to BlockLenUp{c in C, d in D}:
    sum{h in H} x[d,h,c] <= sum{h in H} s[d,h,c] * UpBlock[c];

subject to BlockStarts{c in C, d in D}:
    sum{h in H} s[d,h,c] <= 1;

subject to BlockAdj0{c in C, d in D}:
    x[d,first(H),c] = s[d,first(H),c];
subject to BlockAdj{c in C, d in D, h in H: ord(h) > 1}:
    x[d,h,c] <= s[d,h,c] + x[d,prev(h),c];

subject to OneExam{c in C}:
    sum{d in D, h in H} es[d,h,c] <= 1;

subject to ExamBlock0{c in C, d in D}:
    e[d,first(H),c] = es[d,first(H),c];
subject to ExamBlock{c in C, d in D, h in H: ord(h, H) > 1}:
    e[d,h,c] <= es[d,h,c] + e[d,prev(h),c];

subject to ExamAfterLecture{c in C, d in D, h in H}:
    s[d,h,c] <= sum{h1 in H, d1 in D: ord(d1, D) > ord(d, D)} es[d1,h1,c];

subject to OneExamDay{(d,k) in DK}:
    sum{h in H, (c,k) in CK} es[d, h, c] <= 1;


subject to LectureNights{(fr,sa) in WEEKENDS, l in L}:
    sum{(c,l) in CL} (x[fr,last(H),c] + e[fr,last(H),c]
                      + x[sa,first(H),c] + e[sa,first(H),c]) <= 1;

subject to EnsureUsedWeekends{(sa,so) in WEEKENDS, k in K}:
    sum{h in H} (day_s[k, sa, h] + day_s[k, so, h])
        <= 2*usedWeekend[sa,so, k];

subject to OnlyOnDK{k in K, (d,k) in ({D,K} diff DK),
    (c,k) in CK, h in H}:
        x[d,h,c] = 0;
subject to OnlyOnDKExams{k in K, (d,k) in ({D,K} diff DK),
    (c,k) in CK, h in H}:
        e[d,h,c] = 0;

# only 1 full block each day
subject to DayBlockStarts{k in K, d in D}:
    sum{h in H} day_s[k, d, h] <= 1;

subject to DayBlockAdj0{k in K, d in D}:
    sum{(c,k) in CK} (x[d, first(H), c] + e[d, first(H), c])
        = day_s[k, d, first(H)];

subject to DayBlockAdj{k in K, d in D, h in H: ord(h, H) > 1}:
    sum{(c,k) in CK} (x[d, h, c] + e[d, h, c]) <= day_s[k, d, h]
        + sum{(c,k) in CK} (x[d, prev(h), c] + e[d, prev(h), c]);
```

```
subject to WeekFull{(d,k) in WEEK}:
    sum{h in H, (c,k) in CK} (x[d, h, c] + e[d,h,c]) >= 8;
```

A.3. Heuristic Framework

The heuristics are implemented using the Python programming language, version 2. For access to the databse the SQLAlchemy SQL toolkit and Object Relationship Mapper is used. The genetic algorithms are implemented using the DEAP framework [30].

All the low-level heuristics implement a fixed interface containing a `step` method, such that they can be easily exchanged. The generic algorithms for classic construction (Algorithm 4.1) and for time fill algorithms are implemented and concrete versions are generated using the Mixin concept in Python. This allows an easy adaption of the implementation to new similar problems and the extension with additional low-level heuristics. Also different meta-heuristics could be added easily.

The low-level heuristics are included in implementations of the C02-heuristic, the selection hyper-heuristic and the multi-objective selection heuristic.

B. A concrete Input Example for the CAMPUS 02 Timetabling Problem

In this appendix a concrete example of a practical input to the problem as needed by the problem definition in Section 2.1 is given. It is used for reference, in the content of the thesis. This instance of the problem is also referred to with 'winter term 2015/16'.

- Set of days usable for planning

$$\begin{aligned}D := \{&2015\text{-}09\text{-}18, 2015\text{-}09\text{-}19, 2015\text{-}09\text{-}21, 2015\text{-}09\text{-}22, 2015\text{-}09\text{-}23, 2015\text{-}09\text{-}24,\\ &2015\text{-}09\text{-}25, 2015\text{-}09\text{-}26, 2015\text{-}10\text{-}02, 2015\text{-}10\text{-}03, 2015\text{-}10\text{-}05, 2015\text{-}10\text{-}06,\\ &2015\text{-}10\text{-}07, 2015\text{-}10\text{-}08, 2015\text{-}10\text{-}09, 2015\text{-}10\text{-}10, 2015\text{-}10\text{-}12, 2015\text{-}10\text{-}13,\\ &2015\text{-}10\text{-}14, 2015\text{-}10\text{-}15, 2015\text{-}10\text{-}16, 2015\text{-}10\text{-}17, 2015\text{-}10\text{-}30, 2015\text{-}10\text{-}31,\\ &2015\text{-}11\text{-}06, 2015\text{-}11\text{-}07, 2015\text{-}11\text{-}13, 2015\text{-}11\text{-}14, 2015\text{-}11\text{-}16, 2015\text{-}11\text{-}17,\\ &2015\text{-}11\text{-}18, 2015\text{-}11\text{-}19, 2015\text{-}11\text{-}20, 2015\text{-}11\text{-}21, 2015\text{-}11\text{-}27, 2015\text{-}11\text{-}28,\\ &2015\text{-}12\text{-}04, 2015\text{-}12\text{-}05, 2015\text{-}12\text{-}11, 2015\text{-}12\text{-}12, 2016\text{-}01\text{-}08, 2016\text{-}01\text{-}09,\\ &2016\text{-}01\text{-}15, 2016\text{-}01\text{-}16, 2016\text{-}01\text{-}22, 2016\text{-}01\text{-}23, 2016\text{-}01\text{-}29, 2016\text{-}01\text{-}30,\\ &2016\text{-}02\text{-}05, 2016\text{-}02\text{-}06\}.\end{aligned}$$

- Set of classes

$$K := \{1.\ \text{term BSc}, 2.\ \text{term BSc}, 3.\ \text{term BSc}, 1.\ \text{term MSc}\}.$$

- The lecturers

Lecturer	Courses	NoGo-days
L#61	GML, PJM	
L#62	MA1, WST	
L#63	AKIT1, BA1, BPJ1, DLM/WEW, GMG, WIA	
L#64	BPJ1, DG1, DG2, PM1, BSc	2015-10-02, 2015-10-03
L#65	DLM/WEW, IEG, SSY	2015-10-02, 2015-10-03
L#67	IFM/OR, MA1	2015-09-18, 2015-10-02, 2015-10-03, 2015-10-09, 2015-11-14, 2016-01-08, 2016-01-09, 2016-01-22, 2016-01-23

L#69	NT2	2015-09-18, 2015-09-19, 2015-09-25, 2015-09-26, 2015-10-02, 2015-10-03, 2015-10-30, 2015-10-31, 2015-12-11, 2015-12-12, 2016-01-08, 2016-01-09
L#70	NT2	2016-01-29, 2016-01-30
L#71	INF	
L#73	PJM	
L#76	IFM/OR	
L#77	CON	2015-09-18, 2015-09-25, 2015-11-13, 2015-12-04, 2015-12-11, 2016-01-15, 2016-01-16, 2016-01-29, 2016-01-30, 2016-02-05, 2016-02-06
L#78	GML	
L#79	PCT/SWD	
L#81	KRY	2015-09-18, 2015-09-19, 2015-09-21, 2015-09-22, 2015-09-23, 2015-09-24, 2015-09-25, 2015-09-26, 2015-10-30, 2015-10-31, 2015-11-20, 2015-11-21
L#83	DG1, DG2	
L#85	GEN, NCD	2015-09-21, 2015-10-02, 2015-10-06, 2015-11-17, 2015-11-27, 2016-01-23, 2016-02-05
L#86	BE2	2015-10-12, 2015-10-13, 2015-10-15, 2015-10-17, 2015-11-14, 2015-11-19, 2015-11-28, 2015-12-12
L#87	CM1, KOM, PRT	
L#90	IEG	
L#93	PR1, SWE	2015-09-18, 2015-09-19, 2015-11-06, 2015-11-07, 2015-12-04, 2015-12-05, 2016-01-22, 2016-01-23
L#95	NCD	2015-10-12, 2015-10-13, 2015-10-15, 2015-10-17, 2015-11-14, 2015-11-19, 2015-11-28, 2015-12-12
L#101	SWE	2015-10-12, 2015-10-13, 2015-10-14, 2015-10-15, 2015-10-16, 2015-10-17, 2015-10-30, 2015-10-31
L#129	PR1	
L#130	WEB	

L#131	BUB, IFM/OR	
L#132	BUB	
L#133	KOM, PRT	
L#134	KOM	
L#136	DB1	2015-10-02, 2015-10-03, 2015-10-05, 2015-10-30, 2015-10-31
L#137	BPJ1, PM1	2015-09-25, 2015-09-26, 2015-10-14, 2015-10-15
L#138	BE2, GEN	2015-09-21, 2015-09-22, 2015-09-23, 2015-09-24, 2015-10-02, 2015-10-05, 2015-10-06, 2015-10-07, 2015-10-08, 2015-10-12, 2015-10-13, 2015-10-14, 2015-10-15, 2015-10-16, 2015-11-16, 2015-11-17, 2015-11-18, 2015-11-19, 2015-12-12, 2016-01-23, 2016-02-05
L#139	PRT	
L#140	IDM	
L#141	PCT/SWD, QMG	
L#142	DLM/WEW	
L#143	ESE	
L#144	BPJ1	
L#145	SIM	2015-09-18, 2015-09-19, 2015-09-21, 2015-09-22, 2015-09-23, 2015-09-24, 2015-09-25, 2015-09-26, 2015-10-02, 2015-10-03, 2015-10-05, 2015-10-06, 2015-10-07, 2015-10-08, 2015-10-09, 2015-10-10, 2015-10-12, 2015-10-13, 2015-10-14, 2015-10-15, 2015-10-16, 2015-12-04, 2016-01-08
L#146	SQA	2015-10-12, 2015-11-16, 2015-11-21

- The curriculum

Course c	Class	$CourseHours(c)$	$ExamHours(c)$	$l(c)$	$u(c)$
AKIT1	1. term MSc	26	2	2	4
BPJ1	1. term MSc	15	0	2	9
CM1	1. term MSc	26	0	8	9
ESE	1. term MSc	25	3	3	4
GML	1. term MSc	23	4	3	4
IEG	1. term MSc	30	4	3	4
IFM/OR	1. term MSc	18	2	3	4
NCD	1. term MSc	23	5	3	6
SIM	1. term MSc	26	2	2	4
SQA	1. term MSc	27	2	3	4
BUB	1. term BSc	26	2	4	6
GMG	1. term BSc	25	2	3	4
INF	1. term BSc	29	2	2	5
KOM	1. term BSc	17	0	8	9
MA1	1. term BSc	27	2	3	4
PJM	1. term BSc	27	2	9	9
PR1	1. term BSc	39	3	3	5
WEB	1. term BSc	28	2	2	4
BSc	1. term BSc	23	2	2	4
BPJ1	3. term BSc	6	0	2	4
CON	3. term BSc	29	2	2	4
DB1	3. term BSc	44	2	2	4
DG1	3. term BSc	23	2	2	4
GEN	3. term BSc	25	2	3	6
NT2	3. term BSc	27	2	2	5
PM1	3. term BSc	23	2	2	4

PRT	3. term BSc	17	0	8	9
SWE	3. term BSc	27	1	2	4
WST	3. term BSc	13	2	2	4
BA1	5. term BSc	13	0	2	9
BE2	5. term BSc	26	2	3	6
DG2	5. term BSc	23	2	2	4
DLM/WEW	5. term BSc	25	2	2	4
IDM	5. term BSc	27	2	2	4
KRY	5. term BSc	27	2	2	4
PCT/SWD	5. term BSc	26	2	2	4
QMG	5. term BSc	26	2	2	4
SSY	5. term BSc	23	3	3	4
WIA	5. term BSc	15	0	2	3

- Intensive weeks

Class	Intensive week
1. term BSc	2015-09-21, 2015-09-22, 2015-09-23, 2015-09-24
3. term BSc	2015-10-05, 2015-10-06, 2015-10-07, 2015-10-08
5. term BSc	2015-10-12, 2015-10-13, 2015-10-14, 2015-10-15
1. term MSc	2015-11-16, 2015-11-17, 2015-11-18, 2015-11-19

- Preassigned time slots

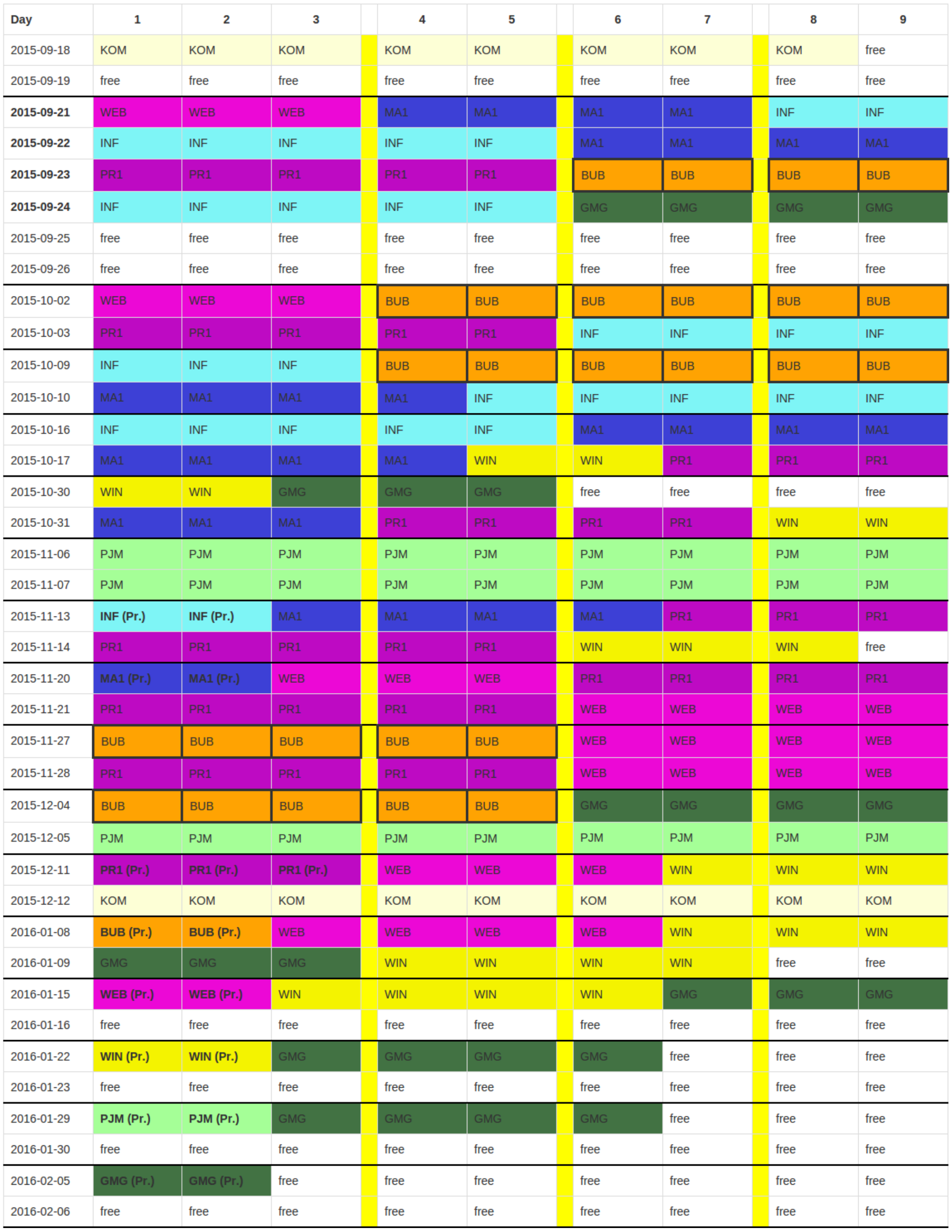
Course	Fixed time slots
BUB	(2015-09-23, 6), (2015-09-23, 7), (2015-09-23, 8), (2015-09-23, 9), (2015-10-02, 4), (2015-10-02, 5), (2015-10-02, 6), (2015-10-02, 7), (2015-10-02, 8), (2015-10-02, 9), (2015-10-09, 4), (2015-10-09, 5), (2015-10-09, 6), (2015-10-09, 7), (2015-10-09, 8), (2015-10-09, 9), (2015-11-27, 1), (2015-11-27, 2), (2015-11-27, 3), (2015-11-27, 4), (2015-11-27, 5), (2015-12-04, 1), (2015-12-04, 2), (2015-12-04, 3), (2015-12-04, 4), (2015-12-04, 5)
CM1	(2015-10-16, 1), (2015-10-16, 2), (2015-10-16, 3), (2015-10-16, 4), (2015-10-16, 5), (2015-10-16, 6), (2015-10-16, 7), (2015-10-16, 8), (2015-10-16, 9), (2015-10-17, 1), (2015-10-17, 2), (2015-10-17, 3), (2015-10-17, 4), (2015-10-17, 5), (2015-10-17, 6), (2015-10-17, 7), (2015-10-17, 8), (2015-10-17, 9), (2015-11-21, 1), (2015-11-21, 2), (2015-11-21, 3), (2015-11-21, 4), (2015-11-21, 5), (2015-11-21, 6), (2015-11-21, 7), (2015-11-21, 8)
PRT	(2015-11-13, 1), (2015-11-13, 2), (2015-11-13, 3), (2015-11-13, 4), (2015-11-13, 5), (2015-11-13, 6), (2015-11-13, 7), (2015-11-13, 8), (2015-11-13, 9), (2015-11-14, 1), (2015-11-14, 2), (2015-11-14, 3), (2015-11-14, 4), (2015-11-14, 5), (2015-11-14, 6), (2015-11-14, 7), (2015-11-14, 8)

C. A Solution obtained with the Selection Hyper-Heuristic

This appendix shows a solution obtained by the selection hyper-heuristic from Section 4.3.2. This solution is already referenced in Section 5.2. The following table again summarizes several important properties of this solution. The class timetables of this solution are shown in Figures C.1, C.2, C.3 and C.4.

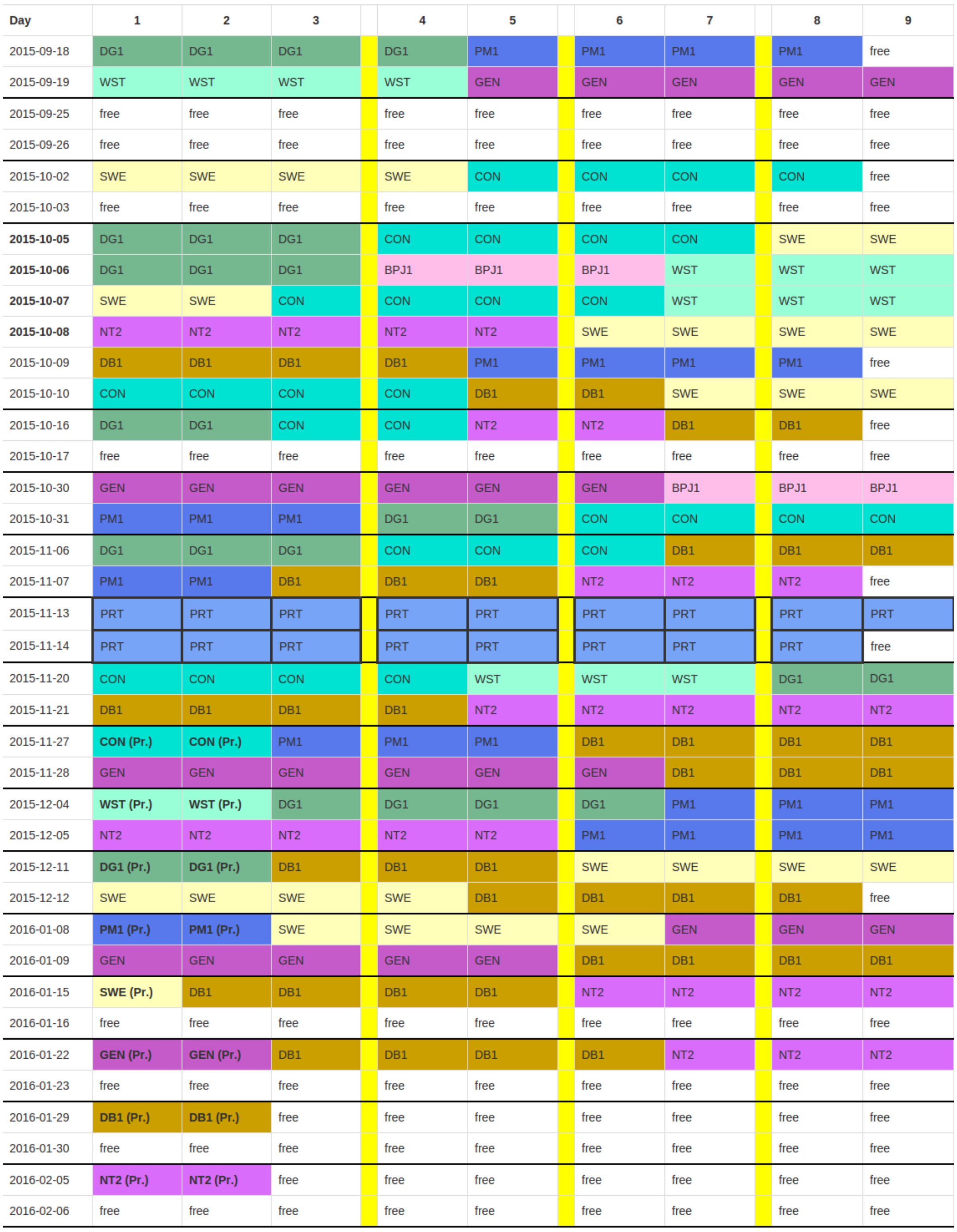
Input data	winter term 2015/16 (see Appendix B)
Heuristic	Best individual from selection hyper-heuristic using heuristic set **H1**.
Running time	13:10,73
Objective (Eq.5.1)	100
Lecturer Objective (Eq. 5.2)	408

		Objectives	
Class	usedWeekends	unusedHours	freePeriods
1. term BSc	16	19	0
3. term BSc	16	14	0
5. term BSc	14	0	0
1. term MSc	15	6	0



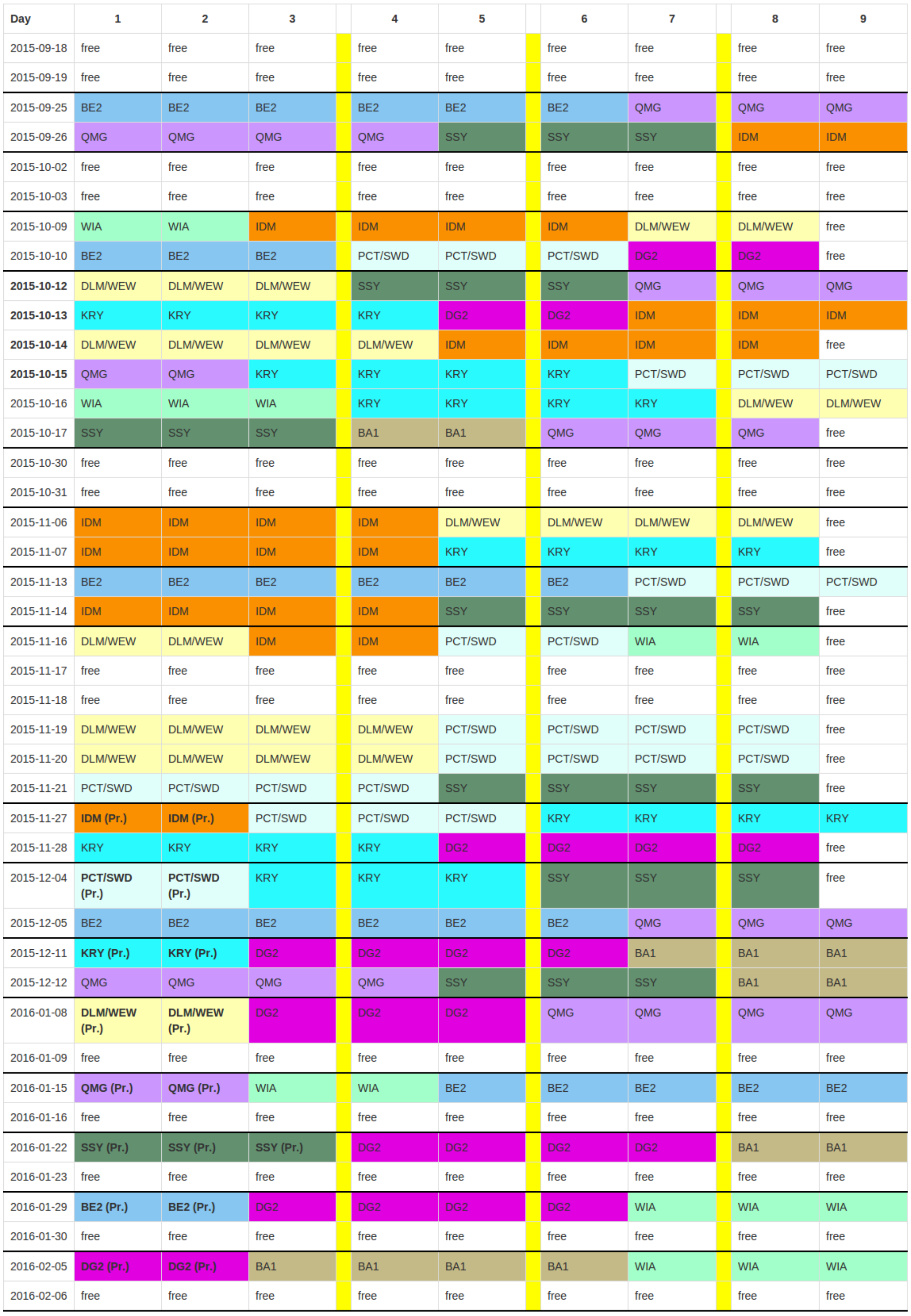
Day	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2015-09-18	KOM	KOM	KOM	KOM	KOM	KOM	KOM	KOM	free
2015-09-19	free	free	free	free	free	free	free	free	free
2015-09-21	WEB	WEB	WEB	MA1	MA1	MA1	MA1	INF	INF
2015-09-22	INF	INF	INF	INF	INF	MA1	MA1	MA1	MA1
2015-09-23	PR1	PR1	PR1	PR1	PR1	BUB	BUB	BUB	BUB
2015-09-24	INF	INF	INF	INF	INF	GMG	GMG	GMG	GMG
2015-09-25	free	free	free	free	free	free	free	free	free
2015-09-26	free	free	free	free	free	free	free	free	free
2015-10-02	WEB	WEB	WEB	BUB	BUB	BUB	BUB	BUB	BUB
2015-10-03	PR1	PR1	PR1	PR1	PR1	INF	INF	INF	INF
2015-10-09	INF	INF	INF	BUB	BUB	BUB	BUB	BUB	BUB
2015-10-10	MA1	MA1	MA1	MA1	INF	INF	INF	INF	INF
2015-10-16	INF	INF	INF	INF	INF	MA1	MA1	MA1	MA1
2015-10-17	MA1	MA1	MA1	MA1	WIN	WIN	PR1	PR1	PR1
2015-10-30	WIN	WIN	GMG	GMG	GMG	free	free	free	free
2015-10-31	MA1	MA1	MA1	PR1	PR1	PR1	PR1	WIN	WIN
2015-11-06	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM
2015-11-07	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM
2015-11-13	**INF (Pr.)**	**INF (Pr.)**	MA1	MA1	MA1	MA1	PR1	PR1	PR1
2015-11-14	PR1	PR1	PR1	PR1	PR1	WIN	WIN	WIN	free
2015-11-20	**MA1 (Pr.)**	**MA1 (Pr.)**	WEB	WEB	WEB	PR1	PR1	PR1	PR1
2015-11-21	PR1	PR1	PR1	PR1	PR1	WEB	WEB	WEB	WEB
2015-11-27	BUB	BUB	BUB	BUB	BUB	WEB	WEB	WEB	WEB
2015-11-28	PR1	PR1	PR1	PR1	PR1	WEB	WEB	WEB	WEB
2015-12-04	BUB	BUB	BUB	BUB	BUB	GMG	GMG	GMG	GMG
2015-12-05	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM	PJM
2015-12-11	**PR1 (Pr.)**	**PR1 (Pr.)**	**PR1 (Pr.)**	WEB	WEB	WEB	WIN	WIN	WIN
2015-12-12	KOM	KOM	KOM	KOM	KOM	KOM	KOM	KOM	KOM
2016-01-08	**BUB (Pr.)**	**BUB (Pr.)**	WEB	WEB	WEB	WEB	WIN	WIN	WIN
2016-01-09	GMG	GMG	GMG	WIN	WIN	WIN	WIN	free	free
2016-01-15	**WEB (Pr.)**	**WEB (Pr.)**	WIN	WIN	WIN	WIN	GMG	GMG	GMG
2016-01-16	free	free	free	free	free	free	free	free	free
2016-01-22	**WIN (Pr.)**	**WIN (Pr.)**	GMG	GMG	GMG	GMG	free	free	free
2016-01-23	free	free	free	free	free	free	free	free	free
2016-01-29	**PJM (Pr.)**	**PJM (Pr.)**	GMG	GMG	GMG	GMG	free	free	free
2016-01-30	free	free	free	free	free	free	free	free	free
2016-02-05	**GMG (Pr.)**	**GMG (Pr.)**	free	free	free	free	free	free	free
2016-02-06	free	free	free	free	free	free	free	free	free

Figure C.1.: Timetable for class ‘1. term BSc’.



Day	1	2	3	4	5	6	7	8	9
2015-09-18	DG1	DG1	DG1	DG1	PM1	PM1	PM1	PM1	free
2015-09-19	WST	WST	WST	WST	GEN	GEN	GEN	GEN	GEN
2015-09-25	free	free	free	free	free	free	free	free	free
2015-09-26	free	free	free	free	free	free	free	free	free
2015-10-02	SWE	SWE	SWE	SWE	CON	CON	CON	CON	free
2015-10-03	free	free	free	free	free	free	free	free	free
2015-10-05	DG1	DG1	DG1	CON	CON	CON	CON	SWE	SWE
2015-10-06	DG1	DG1	DG1	BPJ1	BPJ1	BPJ1	WST	WST	WST
2015-10-07	SWE	SWE	CON	CON	CON	CON	WST	WST	WST
2015-10-08	NT2	NT2	NT2	NT2	NT2	SWE	SWE	SWE	SWE
2015-10-09	DB1	DB1	DB1	DB1	PM1	PM1	PM1	PM1	free
2015-10-10	CON	CON	CON	CON	DB1	DB1	SWE	SWE	SWE
2015-10-16	DG1	DG1	CON	CON	NT2	NT2	DB1	DB1	free
2015-10-17	free	free	free	free	free	free	free	free	free
2015-10-30	GEN	GEN	GEN	GEN	GEN	GEN	BPJ1	BPJ1	BPJ1
2015-10-31	PM1	PM1	PM1	DG1	DG1	CON	CON	CON	CON
2015-11-06	DG1	DG1	DG1	CON	CON	CON	DB1	DB1	DB1
2015-11-07	PM1	PM1	DB1	DB1	DB1	NT2	NT2	NT2	free
2015-11-13	PRT	PRT	PRT	PRT	PRT	PRT	PRT	PRT	PRT
2015-11-14	PRT	PRT	PRT	PRT	PRT	PRT	PRT	PRT	free
2015-11-20	CON	CON	CON	CON	WST	WST	WST	DG1	DG1
2015-11-21	DB1	DB1	DB1	DB1	NT2	NT2	NT2	NT2	NT2
2015-11-27	**CON (Pr.)**	**CON (Pr.)**	PM1	PM1	PM1	DB1	DB1	DB1	DB1
2015-11-28	GEN	GEN	GEN	GEN	GEN	GEN	DB1	DB1	DB1
2015-12-04	**WST (Pr.)**	**WST (Pr.)**	DG1	DG1	DG1	DG1	PM1	PM1	PM1
2015-12-05	NT2	NT2	NT2	NT2	NT2	PM1	PM1	PM1	PM1
2015-12-11	**DG1 (Pr.)**	**DG1 (Pr.)**	DB1	DB1	DB1	SWE	SWE	SWE	SWE
2015-12-12	SWE	SWE	SWE	SWE	DB1	DB1	DB1	DB1	free
2016-01-08	**PM1 (Pr.)**	**PM1 (Pr.)**	SWE	SWE	SWE	SWE	GEN	GEN	GEN
2016-01-09	GEN	GEN	GEN	GEN	GEN	DB1	DB1	DB1	DB1
2016-01-15	**SWE (Pr.)**	DB1	DB1	DB1	DB1	NT2	NT2	NT2	NT2
2016-01-16	free	free	free	free	free	free	free	free	free
2016-01-22	**GEN (Pr.)**	**GEN (Pr.)**	DB1	DB1	DB1	DB1	NT2	NT2	NT2
2016-01-23	free	free	free	free	free	free	free	free	free
2016-01-29	**DB1 (Pr.)**	**DB1 (Pr.)**	free	free	free	free	free	free	free
2016-01-30	free	free	free	free	free	free	free	free	free
2016-02-05	**NT2 (Pr.)**	**NT2 (Pr.)**	free	free	free	free	free	free	free
2016-02-06	free	free	free	free	free	free	free	free	free

Figure C.2.: Timetable for class ‘3. term BSc’.



Day	1	2	3		4	5		6	7		8	9
2015-09-18	free	free	free		free	free		free	free		free	free
2015-09-19	free	free	free		free	free		free	free		free	free
2015-09-25	BE2	BE2	BE2		BE2	BE2		BE2	QMG		QMG	QMG
2015-09-26	QMG	QMG	QMG		QMG	SSY		SSY	SSY		IDM	IDM
2015-10-02	free	free	free		free	free		free	free		free	free
2015-10-03	free	free	free		free	free		free	free		free	free
2015-10-09	WIA	WIA	IDM		IDM	IDM		IDM	DLM/WEW		DLM/WEW	free
2015-10-10	BE2	BE2	BE2		PCT/SWD	PCT/SWD		PCT/SWD	DG2		DG2	free
2015-10-12	DLM/WEW	DLM/WEW	DLM/WEW		SSY	SSY		SSY	QMG		QMG	QMG
2015-10-13	KRY	KRY	KRY		KRY	DG2		DG2	IDM		IDM	IDM
2015-10-14	DLM/WEW	DLM/WEW	DLM/WEW		DLM/WEW	IDM		IDM	IDM		IDM	free
2015-10-15	QMG	QMG	KRY		KRY	KRY		KRY	PCT/SWD		PCT/SWD	PCT/SWD
2015-10-16	WIA	WIA	WIA		KRY	KRY		KRY	KRY		DLM/WEW	DLM/WEW
2015-10-17	SSY	SSY	SSY		BA1	BA1		QMG	QMG		QMG	free
2015-10-30	free	free	free		free	free		free	free		free	free
2015-10-31	free	free	free		free	free		free	free		free	free
2015-11-06	IDM	IDM	IDM		IDM	DLM/WEW		DLM/WEW	DLM/WEW		DLM/WEW	free
2015-11-07	IDM	IDM	IDM		IDM	KRY		KRY	KRY		KRY	free
2015-11-13	BE2	BE2	BE2		BE2	BE2		BE2	PCT/SWD		PCT/SWD	PCT/SWD
2015-11-14	IDM	IDM	IDM		IDM	SSY		SSY	SSY		SSY	free
2015-11-16	DLM/WEW	DLM/WEW	IDM		IDM	PCT/SWD		PCT/SWD	WIA		WIA	free
2015-11-17	free	free	free		free	free		free	free		free	free
2015-11-18	free	free	free		free	free		free	free		free	free
2015-11-19	DLM/WEW	DLM/WEW	DLM/WEW		DLM/WEW	PCT/SWD		PCT/SWD	PCT/SWD		PCT/SWD	free
2015-11-20	DLM/WEW	DLM/WEW	DLM/WEW		DLM/WEW	PCT/SWD		PCT/SWD	PCT/SWD		PCT/SWD	free
2015-11-21	PCT/SWD	PCT/SWD	PCT/SWD		PCT/SWD	SSY		SSY	SSY		SSY	free
2015-11-27	**IDM (Pr.)**	**IDM (Pr.)**	PCT/SWD		PCT/SWD	PCT/SWD		KRY	KRY		KRY	KRY
2015-11-28	KRY	KRY	KRY		KRY	DG2		DG2	DG2		DG2	free
2015-12-04	**PCT/SWD (Pr.)**	**PCT/SWD (Pr.)**	KRY		KRY	KRY		SSY	SSY		SSY	free
2015-12-05	BE2	BE2	BE2		BE2	BE2		BE2	QMG		QMG	QMG
2015-12-11	**KRY (Pr.)**	**KRY (Pr.)**	DG2		DG2	DG2		DG2	BA1		BA1	BA1
2015-12-12	QMG	QMG	QMG		QMG	SSY		SSY	SSY		BA1	BA1
2016-01-08	**DLM/WEW (Pr.)**	**DLM/WEW (Pr.)**	DG2		DG2	DG2		QMG	QMG		QMG	QMG
2016-01-09	free	free	free		free	free		free	free		free	free
2016-01-15	**QMG (Pr.)**	**QMG (Pr.)**	WIA		WIA	BE2		BE2	BE2		BE2	BE2
2016-01-16	free	free	free		free	free		free	free		free	free
2016-01-22	**SSY (Pr.)**	**SSY (Pr.)**	**SSY (Pr.)**		DG2	DG2		DG2	DG2		BA1	BA1
2016-01-23	free	free	free		free	free		free	free		free	free
2016-01-29	**BE2 (Pr.)**	**BE2 (Pr.)**	DG2		DG2	DG2		DG2	WIA		WIA	WIA
2016-01-30	free	free	free		free	free		free	free		free	free
2016-02-05	**DG2 (Pr.)**	**DG2 (Pr.)**	BA1		BA1	BA1		BA1	WIA		WIA	WIA
2016-02-06	free	free	free		free	free		free	free		free	free

Figure C.3.: Timetable for class ‘3. term BSc’.

Day	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2015-09-18	free	free	free	free	free	free	free	free	free
2015-09-19	free	free	free	free	free	free	free	free	free
2015-09-25	NCD	NCD	NCD	NCD	NCD	NCD	AKIT1	AKIT1	AKIT1
2015-09-26	IEG	IEG	IEG	IEG	GML	GML	GML	AKIT1	AKIT1
2015-10-02	SQA	SQA	SQA	ESE	ESE	ESE	ESE	free	free
2015-10-03	NCD	NCD	NCD	NCD	NCD	NCD	GML	GML	GML
2015-10-09	free	free	free	free	free	free	free	free	free
2015-10-10	free	free	free	free	free	free	free	free	free
2015-10-16	CM1	CM1	CM1	CM1	CM1	CM1	CM1	CM1	CM1
2015-10-17	CM1	CM1	CM1	CM1	CM1	CM1	CM1	CM1	CM1
2015-10-30	IEG	IEG	IEG	IEG	GML	GML	GML	GML	free
2015-10-31	SIM	SIM	SIM	SIM	AKIT1	AKIT1	AKIT1	AKIT1	free
2015-11-06	IFM/OR	IFM/OR	IFM/OR	SQA	SQA	SQA	ESE	ESE	ESE
2015-11-07	SIM	SIM	SIM	AKIT1	AKIT1	AKIT1	NCD	NCD	NCD
2015-11-13	SQA	SQA	SQA	IEG	IEG	IEG	IEG	AKIT1	AKIT1
2015-11-14	AKIT1	AKIT1	GML	GML	GML	ESE	ESE	ESE	free
2015-11-16	NCD	NCD	NCD	NCD	NCD	SIM	SIM	SIM	SIM
2015-11-17	SIM	SIM	SIM	SIM	SQA	SQA	SQA	SQA	free
2015-11-18	NCD	NCD	NCD	SIM	SIM	SIM	SQA	SQA	SQA
2015-11-19	IFM/OR	IFM/OR	IFM/OR	SIM	SIM	SIM	SIM	AKIT1	AKIT1
2015-11-20	SIM	SIM	SIM	SIM	AKIT1	AKIT1	AKIT1	AKIT1	free
2015-11-21	CM1	CM1	CM1	CM1	CM1	CM1	CM1	CM1	free
2015-11-27	**SIM (Pr.)**	**SIM (Pr.)**	AKIT1	AKIT1	AKIT1	AKIT1	SQA	SQA	SQA
2015-11-28	SQA	SQA	SQA	SQA	ESE	ESE	ESE	ESE	free
2015-12-04	**AKIT1 (Pr.)**	**AKIT1 (Pr.)**	IEG	IEG	IEG	SQA	SQA	SQA	SQA
2015-12-05	IFM/OR	IFM/OR	IFM/OR	IFM/OR	ESE	ESE	ESE	ESE	free
2015-12-11	**NCD (Pr.)**	**NCD (Pr.)**	**NCD (Pr.)**	**NCD (Pr.)**	**NCD (Pr.)**	IFM/OR	IFM/OR	IFM/OR	IFM/OR
2015-12-12	IEG	IEG	IEG	IEG	IFM/OR	IFM/OR	IFM/OR	IFM/OR	free
2016-01-08	**SQA (Pr.)**	**SQA (Pr.)**	GML	GML	GML	IEG	IEG	IEG	IEG
2016-01-09	ESE	ESE	ESE	ESE	GML	GML	GML	free	free
2016-01-15	**IFM/OR (Pr.)**	**IFM/OR (Pr.)**	ESE	ESE	ESE	GML	GML	GML	GML
2016-01-16	free	free	free	free	free	free	free	free	free
2016-01-22	**GML (Pr.)**	**GML (Pr.)**	**GML (Pr.)**	**GML (Pr.)**	BPJ1	BPJ1	BPJ1	BPJ1	free
2016-01-23	IEG	IEG	IEG	IEG	BPJ1	BPJ1	BPJ1	free	free
2016-01-29	**ESE (Pr.)**	**ESE (Pr.)**	**ESE (Pr.)**	IEG	IEG	IEG	BPJ1	BPJ1	BPJ1
2016-01-30	free	free	free	free	free	free	free	free	free
2016-02-05	**IEG (Pr.)**	**IEG (Pr.)**	**IEG (Pr.)**	**IEG (Pr.)**	BPJ1	BPJ1	BPJ1	BPJ1	BPJ1
2016-02-06	free	free	free	free	free	free	free	free	free

Figure C.4.: Timetable for class '1. term MSc'.