Application of Mixed Integer Programming Methods for Practical Educational Timetabling

Mikkelsen, Rasmus Ørnstrup

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

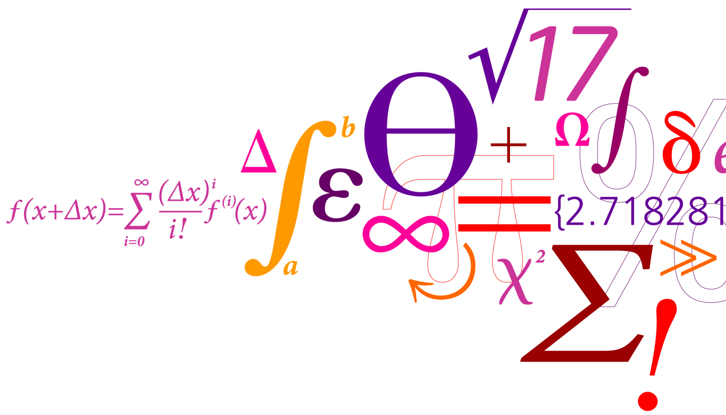
Citation (APA):
Mikkelsen, R. Ø. (2021). *Application of Mixed Integer Programming Methods for Practical Educational Timetabling*.

Application of Mixed Integer Programming Methods for Practical Educational Timetabling

Ph.D. Thesis

Rasmus Ørnstrup Mikkelsen

August, 2021



Title:	Application of Mixed Integer Programming Methods for Practical Educational Timetabling
Type:	Ph.D. Thesis
Date:	August, 2021

Author:	Rasmus Ørnstrup Mikkelsen
Supervisors:	Associate Professor Thomas J. R. Stidsen Management Science DTU Management Technical University of Denmark Akademivej, Building 358 DK-2800 Kgs. Lyngby
	Matias Sørensen MaCom A/S Vesterbrogade 48, 1. DK-1620 København V
University:	Technical University of Denmark
Department:	Technology, Management and Economics
Division:	Management Science
Address:	Akademivej, Building 358 DK-2800 Kgs. Lyngby
Telephone:	+45 45254800

Abstract

Educational institutions such as high schools and universities face many challenging timetabling problems. Timetables are essential for ensuring that students and instructors can attend their required courses and examinations. Creating timetables that satisfy all requirements is challenging, and it becomes even more challenging when introducing preferences. Thus, timetabling personnel often use tools that assist or automatically create timetables. Much scientific literature since the 1960s has focused on developing such decision support tools and algorithms.

This thesis deals with the application of Mixed Integer Programming (MIP) methods for practical educational timetabling. The literature has focused mainly on the application of heuristics for such problems. Heuristics can generally provide good results in a short amount of time but suffer from being very problem-specific. Additionally, they do not present any information on the quality of the found solutions and depend on external knowledge for evaluation. MIP methods present a mathematical approach that finds solutions and provides means of evaluating solution quality. However, these methods may have difficulties handling immense problems, as is often the case for practical timetabling. This thesis aims to investigate how to apply MIP-based methods for practical educational timetabling successfully.

The thesis is composed of two parts. The first part introduces aspects of practical educational timetabling and the main problem categories considered in the literature. The second part contains the scientific papers produced during this Ph.D. study. The articles apply MIP-based methods to problems originating from practical educational timetabling, specifically, high school semester timetabling and university semester and examination timetabling.

The first article introduces a decision support tool to help practical high school timetabling personnel change a given semester timetable. A timetabling suite used by approximately 200 Danish high schools released the method in February 2020, so the tool is available for practical use. The following four papers (including one technical report) concern the university course timetabling problem presented by the International Timetabling Competition 2019 (ITC 2019). The problem applies to many real-world international universities, and the MIP-based method described in these articles won the competition. The final article presents several solution techniques for a real-world examination timetabling problem from Italian universities. On this problem, the MIP-based approaches yield nearly comparable results with a specialized heuristic.

The articles presented in this thesis collectively validate that MIP-based methods are relevant for practical educational timetabling problems and worthy of more study.

Resumé (Danish Abstract)

Uddannelsesinstitutioner, såsom gymnasier og universiteter, bliver konstant konfronteret med mange udfordrende skemalægningsproblemer. Skemaer er essentielle for at sikre, at studerende og undervisere kan deltage i deres undervisning og eksamener. Det er udfordrende at planlægge skemaer, som overholder alle nødvendige krav, og kompleksiteten øges når der introduceres præferencer til skemaet. Skemalægningspersonale bruger derfor ofte skemalægningsværktøjer, der understøtter eller automatisk lægger skemaer. Siden 1960'erne er der i den videnskabelige litteratur blevet studeret metoder og algoritmer til at understøtte disse opgaver.

Denne afhandling behandler anvendelsen af matematiske programmeringsmetoder (MP), især blandede heltalsprogrammering, på virkelige uddannelsesskemalægningsproblemer. Litteraturen har for disse problemstillinger primært fokuseret på heuristiske løsningsmetoder. Heuristikker kan generelt give gode resultater hurtigt, men har den ulempe, at de er meget problemspecifikke. Derudover giver de ingen information omkring kvaliteten af de løsninger heuristikken genererer og er derfor afhængige af ekstern viden for kvalitetsvurdering. MP-baserede metoder præsenterer en matematisk tilgang, som genererer løsninger og samtidig kan evaluere kvaliteten af løsningerne. Dog kan disse metoder have svært ved at håndtere enorme problemer, som ofte er tilfældet for ikke-teoretiske skemalægningsproblemer. Formålet med denne afhandling er at undersøge, hvordan man med succes kan anvende MP-baserede metoder til praktisk skemalægning.

Afhandlingen består af to dele: Den første del introducerer nogle aspekter af praktisk uddannelsesskemalægning samt hovedkategorierne af de typer af problemer som litteraturen fokuserer på. Den anden del indeholder de videnskabelige artikler, som er udarbejdet i løbet af ph.d.-studiet. Artiklerne præsenterer MP-baserede metoder både til semesterskemalægning i gymnasier og på universiteter samt eksamensplanlægning på universiteter.

Den første artikel introducerer et redskab til at understøtte gymnasieskemalæggere til at indføre ændringer i et allerede lagt skema. Værktøjet blev udgivet i februar 2020 i et skemalægningssystem, som bruges af omkring 200 danske gymnasier og andre ungdomsuddannelser. De næste fire artikler (hvoraf én er en teknisk rapport) omhandler det universitetssemesterskemalægningsproblem, som blev introduceret i den Internationale Skemalægnings Konkurrence 2019. Denne problemformulering kan anvendes på mange universiteter fra den virkelige verden. Den MP-baserede løsningsmetode beskrevet i disse ar-

tikler vandt konkurrencen. Den sidste artikel implementerer og sammenligner flere forskellige løsningsmetoder til et virkeligt eksamensplanlægningsproblem fra italienske universiteter. De MP-baserede metoder præsterer næsten lige så godt som en specialiseret heuristik.

Artiklerne inkluderet i denne afhandling bekræfter, at MP-baserede løsningsmetoder er relevante for praktiske uddannelsesskemalægningsproblemer og fortjener mere videnskabelig opmærksomhed.

Preface

This thesis is part of the requirements for acquiring the degree *Philosophiae Doctor* (Ph.D.) at the Technical University of Denmark. This Ph.D. study was performed under the Industrial Ph.D. Program, in which the Ph.D. candidate is employed in a private company and enrolled at a university. An Industrial Ph.D. should contribute to the company's perspectives and fulfill the requirements of the Ph.D. school. MaCom A/S (the company) and Innovation Fund Denmark (IFD) financially supported the project.

The Ph.D. project has been conducted at the Division of Management Science in the Department of Technology, Management and Economics at the Technical University of Denmark from June 2018 to August 2021. Associate Professor Thomas J. R. Stidsen supervised the project with Matias Sørensen as company supervisor. A substantial part of the studies has been performed jointly with Ph.D. student Dennis Holm from the Department of Technology, Management and Economics, Technical University of Denmark. The thesis consists of an introduction, a technical report, and five papers submitted to peer-reviewed journals.

Kgs. Lyngby, Denmark, August 2021

Rasmus Mikkelsen

Rasmus Ørnstrup Mikkelsen

Acknowledgments

First of all, I would like to thank my supervisors Thomas J. R. Stidsen and Matias Sørensen, for their essential guidance throughout this Ph.D. study. They have provided me with crucial support and input both on an academic and personal level.

I would also like to thank MaCom A/S for being an exceptional workplace. I genuinely believe MaCom is extraordinarily good at embracing its employees and cultivating a productive and enjoyable work environment. I am grateful for the great colleagues who have been interested in me and the project, and I am very thankful for the patience and care shown towards me during difficult times. I would especially like to thank Mads Poulsen, Martin Holbøll, and Michael Herold for giving me the opportunity to work on this project.

Additionally, I would like to thank my colleagues at the division of Management Science of the Department of Technology, Management and Economics of DTU. Many people have provided input and feedback throughout the project.

Thank you to Professor Andrea Schaerf for hosting me on an “external” research stay. Unfortunately, the COVID-19 pandemic made a physical visit impossible, and I appreciate his flexibility regarding remote collaboration.

A special thank you goes to my colleague and friend Dennis S. Holm. Dennis and I have collaborated a lot during our respective Ph.D. projects, and I am proud of what we have accomplished together. I have benefited much from both our academic and very non-academic discussions and am grateful for our many foosball games.

Last but definitely not least, I would like to direct an enormous thank you to my wife Sara for her incredible support throughout the project and challenging times. Thank you for being you and allowing me to be me — dig og mig imod verden.

Contents

Part I

Introduction

Chapter 1

Thesis Background

"If You are planning for a year, sow rice; if you are planning for a decade, plant trees; if you are planning for a lifetime, educate people."

— Chinese Proverb

Education is essential for the continued betterment of humanity. Luckily, in most places around the world, formal education is a natural part of growing up. Formal education typically divides into four distinct stages: preschool, primary school, secondary school (also known as high school), and post-secondary school (e.g., university and vocational school). Most countries even have compulsory education up to the secondary school level. Thereby, most adults have spent a significant portion of their life attending school and are very familiar with an educational timetable.

In educational institutions, the timetable is of significant importance for students, staff, and the institution itself. Students and staff want timetables that "make sense" from their perspective while not hindering them in other parts of their lives. The institution is interested in providing the required teaching while utilizing its resources as best as possible. Therefore, it is vital to create timetables that strike a balance between the needs of students, staff, and the institution.

In the earlier stages of formal education, students learn fundamental subjects and typically have few teachers. Later stages are characterized by more subjects, electives, branching and specialized study directions, and more requirements, making the problem of producing timetables more difficult. Manually developing such timetables is very time-consuming, cumbersome, and is likely to result in unsatisfactory timetables. Therefore, much attention has been devoted to the research area of Educational Timetabling. The educational timetabling literature mainly studies high school timetabling, university course timetabling, and university examination timetabling (Schaerf, 1999; Kingston, 2013). This thesis considers all three sub-areas from a practical, rather than theoretical, perspective but focuses more on university course timetabling.

Researchers often consider timetabling problems to be optimization problems where the goal is to find the best possible solution from the set of all feasible solutions. Optimization problems are an important part of *Operations Research* (OR). OR can be described as "a discipline that deals with the application of advanced analytical methods to help make better decisions" (INFORMS, 2021) and exists in the intersection of computer science and mathematics. By applying OR techniques to educational timetabling problems, practitioners can take advantage of computers to assist in developing practical educational timetables, leading to improved efficiency and better utilization of resources.

1.1 Operations Research

This section gives a basic introduction to OR. Readers who are well versed in OR can skip this section and continue unto Section 1.2 (Why Mixed Integer Programming?).

OR techniques allow for making good, or better yet, optimal decisions for practical decision problems. Thus, OR has a wide range of application areas, like logistics, supply chain management, revenue management, work crew scheduling, production scheduling, design of shipping networks, timetabling, facility locations, to name a few. Researchers use OR methods to analyze the problem, typically using advanced mathematical methods, and determine the best course of action.

Mathematical models composed of variables and constraints can express a decision problem. Variables represent decisions, and constraints impose conditions on decisions. Usually, a problem will contain both hard and soft constraints, where hard constraints model conditions that a solution must respect. If a solution violates any hard constraint, it is deemed infeasible and thereby not valid for the problem. Soft constraints represent preferences for the characteristics of solutions, i.e., solution features that are desirable or unwanted. Often, the problem includes an objective function that measures a given decision's impact, typically using the weighted violation of soft constraints. It is not uncommon for the objective function to be composed of multiple conflicting objectives.

1.1.1 Integer Programming

Integer Programming (IP) represents a common and powerful method within OR. An IP model is a mathematical optimization model where all variables are required to take integer values. An optimization model includes a number of constraints, variables, and an objective function. The goal is to find a solution that satisfies hard constraints (including variable integrality conditions) and minimizes or maximizes the objective function, depending on the problem. If the problem allows for a subset of variables to be continuous, i.e., they have no integrality requirement, then the model is a Mixed Integer Programming (MIP) model. This thesis uses the term MIP even though the given model may be an IP model. Additionally, all models are implicitly understood to only contain linear terms in constraints and objectives, such that this thesis only deals with (mixed) integer linear programs.

By relaxing the integrality conditions of all variables, such that they are continuous variables, the model becomes a Linear Program (LP), more specifically, the LP-relaxation of the model. The LP-relaxation is typically much easier to solve than the original MIP model. Solving a relaxation of a problem provides a bound on the objective value of the optimal solution. Thereby, the bound represents a theoretical limit on the best possible solution for the problem. Consequently, the bound allows for evaluating the quality of a known solution by considering the gap in solution value to the bound. A solution with a value equal to the bound is optimal. Thus, MIP can solve problems to proven optimality, and thereby MIP belongs to the family of exact methods.

MIP models are solved using MIP solvers, which internally make heavy use of LP-relaxations and optimality gaps. An essential technique used by MIP solvers is the branch-and-bound algorithm, introduced by Land and Doig (2010) which decomposes the problem into subproblems to be solved individually. In very simplified terms, the solver solves the LP-relaxation of each subproblem to obtain bounds and feasible solutions, which it uses to cut branches of the search tree that cannot yield better results. While solving a MIP model, the bound is regularly improved and, even if the model is not solved to optimality, the final bound still provides valuable information about the problem and known solutions.

1.1.2 Heuristics

Heuristics represent another type of solution approach for solving optimization problems. A heuristic is an algorithmic approach that applies "rules of thumb" to search for good solutions. Compared to exacts methods like MIP, heuristics are usually much less sensitive to the size of the problem and can often find good solutions much more quickly. However, this comes at the trade-off of optimality, as the heuristic itself typically cannot determine any bounds. Thus, the quality of found solutions can only be evaluated using external information, like bounds provided by exact methods or prior experience in practical situations. Local search is a common heuristic that starts from an initial solution and iteratively attempts to find better solutions. In each iteration, the heuristic explores a neighborhood defined as all solutions obtainable by applying some operation to the current solution. If the search finds an improving solution, the new solution is accepted and used as the current solution in the next iteration. At some point, the search will reach a local optimum, meaning that no improving solutions exist in the neighborhood, and the search terminates.

A heuristic is a problem-dependent solution strategy, and the term metaheuristics (coined by Glover (1986)) denote higher-level, problem-independent heuristic frameworks. Metaheuristics represent general optimization algorithm designs that do not take advantage of problem-specific features, although some problem-dependent parameter tuning may be advisable. There are many different metaheuristics and numerous different categorizations, for example, single solution and population-based frameworks.

Simulated annealing (Kirkpatrick et al., 1983) is an example of a single-solution metaheuristic, which improves upon the local search heuristic by allowing the acceptance of deteriorating solutions in a probabilistic manner to escape a local optimum and widen the search. Tabu search (Glover, 1986) is another local search-based single solution metaheuristic that will accept a worsening solution in lack of improving solutions. Additionally, the search includes a memory of forbidden (tabu) moves (or similar) to deter the search from returning to previously found solutions.

Population-based metaheuristics maintain and improve a population of solutions. One prominent family of population-based metaheuristics is evolutionary algorithms, which mimic biological evolution and natural selection. In such frameworks, the algorithm subjects the population of solutions to random changes called mutations, combines solutions using a crossover operation (simulating sexual reproduction), performs (natural) selection, and eliminates the least fit solutions, allowing the rest to repeat the cycle in the next generation. The discussed methods represent only a tiny fraction of metaheuristics, and researchers actively investigate different metaheuristic design improvements and practical applications.

1.1.3 Matheuristics

An emerging theme is the hybridization of multiple metaheuristics into one unified framework. Such a combination of methods allows for utilizing complementary strengths of each particular method (Blum et al., 2011). Combining heuristics and exact methods results in a class of methods called matheuristics or model-based metaheuristics. The goal of matheuristics is to gain the speed of heuristics while retaining some properties of exact methods like bounding information. Typically, matheuristics will either use a metaheuristic for higher-level control of the exact method or an exact method for higher-level control over heuristics. MIP solvers fall into the latter category, where branch-and-bound is the exact method controlling the search and the solver employs heuristics in the tree nodes to accelerate the search (Lodi, 2013). This thesis predominantly uses the matheuristic Fix-and-Optimize, which falls into the former category. Lindahl et al. (2018) introduced the fix-and-optimize matheuristic applied to university course timetabling and achieved good results. The matheuristic is a local search that uses a MIP model to define and search the neighborhood in each iteration. Fixing some subset of variables limits the search to a smaller part of the total solution space and makes the MIP model more manageable. If the search finds an improving solution, it becomes the starting solution for the next iteration, where it unfixes previously fixed variables before continuing and fixing a new subset of variables.

1.2 Why Mixed Integer Programming?

This section tries to answer the question, "why try to use mixed integer programming methods for educational timetabling?" A valid question, to which this section presents several answers.

Most recent literature on educational timetabling has focused on other methods, primarily metaheuristics, rather than mixed integer programming methods for solving educational timetabling problems (Qu et al., 2009; Pillay, 2010; Kristiansen and Stidsen, 2013; Bettinelli et al., 2015; Vrielink et al., 2016). Bettinelli et al. (2015) provides an overview of curriculum-based course timetabling and explicitly discusses how most works that contribute exact methods focus primarily on producing lower bounds and often have worse performance regarding finding high-quality solutions. However, MIP-based methods have shown promise and even represent the current state-of-the-art for high school timetabling (Tan et al., 2021). Thus, it is relevant to investigate how to apply MIP-based methods in a broader sense (like practical dynamic timetabling) and to examine if MIP-based methods can achieve the same level of success as other methods within other areas of educational timetabling.

Shortly after starting this Ph.D. study, the International Timetabling Competition 2019 was announced, introducing a novel university course timetabling problem. The introduced problem definition is very complex and known to be applicable in many different practical university problems. This problem can become a crucial benchmark problem in the future, which significantly motivates investigating how to apply MIP-based methods to this problem successfully.

Finally, this is an Industrial Ph.D. study, and the research should have relevance to the industrial partner. Together with the industrial partner, we want to investigate the use of MIP-based methods in practice, as metaheuristic approaches are much more common in automated, practical timetabling systems (Vrielink et al., 2016). From an industrial perspective, MIP modeling has some additional benefits. A MIP model gives a concrete description of the problem, and once we have developed the model, it requires little maintenance. For example, we can easily adjust the model to reflect changes in the problem, such as removing or adding constraints or objectives. Many other types of optimization techniques are more problem-dependent, and problem changes can therefore be catastrophic.

An additional benefit from using MIP-based methods in an industrial setting comes from gradual MIP solver improvements. Robert E. Bixby, founder of CPLEX optimization and Gurobi optimization, argues for huge MIP model solve speedups from MIP solver improvements alone (Bixby, 2014). For example, from version 1.2 to 11, CPLEX has seen a 29,530 times speedup for solving MIP models. He further argues that from the year 1990 to 2014, algorithms have yielded improvements of 870,000 times and machines "only" 6,500 times. Thus algorithms have been the most significant contributing factor to the overall net improvement ($870,000 \cdot 6,500 = 5,655,000,000$ total speed up!). Thus, MIP-based methods will see a "free" improvement simply from MIP solver advances, which is valuable for the industrial partner since they can provide product enhancements for their customers without upgrading hardware or developing upon the method or MIP model. We expect the improvement trend to continue and have throughout this Ph.D. study first-hand seen performance gains solely from updating to newer MIP solver versions.

1.3 Thesis Reading Guide

This thesis is divided into two parts. Part I introduces the research area, the problems addressed, scientific contributions, and the work's conclusions.

Chapter 2 introduces the main research area by providing an overview of common educational timetabling problems, focusing on broadly adopted benchmark problems. Chapter 3 outlines the results of the Ph.D. study, including a summary of the scientific papers, contributions, and practical applications of the work. Chapter 4 concludes and discusses possibilities for future research.

Part II constitutes the majority of the thesis and contains the work produced throughout the Ph.D. study, comprised of five articles and one technical report.

Chapter 2

Educational Timetabling

Educational timetabling is the research area dedicated to modeling and solving the problem of timetabling in educational institutions. In practice, timetabling problems are more difficult for high schools and universities than for lower levels of formal education. Thus, educational timetabling research mainly focuses on the timetabling problems arising in these types of institutions. Schaerf (1999) classified educational timetabling problems into three categories: high school timetabling, university course timetabling, and university examination timetabling. Educational timetabling problems may not strictly belong to one specific category, and other planning problems exists, but the literature has used this general categorization since.

A significant number of people are affected by educational timetables. The following text highlights some of the main stakeholder groups. Each group has its wants and needs regarding the different planning activities that occur in educational institutions.

- Students want balanced timetables such that they have no "wasted" time but also without too heavy peak workload. As the number of students is quite large, and their requirements and needs may be varied, it is difficult to determine what constitutes the best timetable for the group. Individual students have little or no direct influence on the timetabling problems.

- Teachers have different desires regarding the educational timetable, i.e., their work schedule. Some prefer their workload compressed into fewer busy days, and others prefer their teaching evenly spread between available days. Consequently, teachers can often declare preferences or place restrictions on timetabling activities.

- The administration wants good and fair timetables for students and teachers and must ensure that all legal requirements are satisfied. The administration is also very interested in having a high utilization of resources to save costs wherever possible. Additionally, the administration is expected to solve or supply the tools for solving the timetabling problems.

In general terms, educational timetabling is the problem of assigning times and resources to events subject to a number of constraints (McCollum et al., 2010; Pillay, 2010). Events are typically meetings between teachers and classes for high school timetabling, lectures for university course timetabling, and exams for university examination timetabling. Resources typically include students, teachers, and rooms. Often, times are considered in discrete timeslots (also called periods), but this is not always the case. The constraints imposed on a given educational timetabling problem depend not only on its appertaining category but also on other external parameters, such as university/high school structure and legal requirements imposed by the respective country.

A central constraint is to avoid resource double-bookings (or conflicts). A double-booking occurs when scheduling a resource to attend two events at the same time. Timetabling problems always include some form of double-booking constraints, but many other constraint types exist. The most common are listed here.

- Time assignment: Events should be assigned an available time. Different aspects determine the availability of times; for example, a teacher required for an event may be unavailable on certain days. It is also possible that events are preassigned to a specific time. Sets of events may also be required to be scheduled simultaneously or in some strict order.

- Resource assignment: Most commonly, the problem includes assigning rooms to events but sometimes also students or teachers. Problems do not always include room assignments or may initially skip it for all or some events. Room assignments must respect the total room capacity.

- Continuity: This constraint type enforces timetable consistency, for example, ensuring that all events belonging to a single course use the same room.

- Compactness: Constraints used to create compact timetables for students and teachers, either reducing idle time or concentrating events to fewer days, allowing for days off.

- Spreading: Used to spread events when it is desirable to do so. For example, it is desirable to spread out student examinations or lectures of the same course.

Constraints can be either hard or soft. Hard constraints must be satisfied to consider a timetable feasible, e.g., person double-booking is a typical hard constraint. Soft constraints must not necessarily be satisfied but define desirable timetable characteristics, e.g., preferred time assignments or compactness or spreading constraints. The optimal timetable then has no hard constraint violations and the minimum possible soft constraint violations. Likewise, when considering multiple feasible timetables, the timetable with the smallest soft constraint violation is best. Soft constraints can be contradictory, and it is often impossible to attain a soft constraint cost of zero (Pillay, 2016).

For both high school timetabling and university course timetabling, the goal is to develop a (generally) weekly timetable for the institution, such that all students and teachers know the time and location of all their subjects/courses. Although both problems share the same goal, they are vastly different from each other due to each institution type's contrasting characteristics and requirements. Carter and Laporte (1998) highlights that the scheduling problem is very tight for high schools, compared to university course timetabling, which allows for more flexibility. For university course timetabling, the difficult part is to ensure that all students can feasibly select courses required by their program. Table 2.1 shows the characteristic differences accentuated by Carter and Laporte (1998).

Characteristic	High School	University
Scheduling	- by classes	- by student
Choice	- few choices - highly structured programs	- many electives - loosely structured program
Teacher availability	- tight (heavy teaching load)	- flexible (light teaching load)
Student load	- very tight (busy all day)	- fairly loose - days and evenings
Rooms	- few rooms - same size - central location	- many rooms - variety of sizes - decentralized
Criteria	- no conflicts	- minimum conflicts

Table 2.1: General characteristic differences between High School and University Course Timetabling problems from Carter and Laporte (1998).

University examination timetabling differs from the two other categories both structurally and by goal. Typically, a course has a final examination to be scheduled after the end of a semester. The purpose of examination timetabling is to assign all exams to a time and (potentially) a room, such that all students can attend their exams. This timetabling problem also includes proximity constraints, including the undesirable aspect of students attending multiple exams close together in time (Kingston, 2013).

Educational timetabling includes other types of relevant timetabling problems, but most literature focuses on the aforementioned three categories (Kingston, 2013). *Student sectioning* represents another noteworthy problem. In cases of large student enrollments, the relevant courses are broken into separate copies of the course, called sections. The problem is then to assign students, times, and rooms to sections, subject to constraints and objectives.

Solving real instances of educational timetabling problems is exceedingly difficult. In fact, the three main educational timetabling problems (and many others) can be proved to be $\mathcal{NP}$-complete by using a reduction from graph coloring (Cooper and Kingston, 1996). Thus, although the research community should consider many different facets of practical timetabling, it is still worth focusing on the core problems.

Educational tabling research has a long history dating back to the 1960s, e.g., Gotlieb (1962), Brandt (1963), and Hansen (1967). However, for most of its history, research has been very fragmented, as research groups have used their own problem definitions and data (Kingston, 2013), resulting in slower progression of the research area. Only by working on common problem definitions and data can researchers qualitatively compare solution approaches, which is essential to evaluating the success of a given technique and improving the state-of-the-art. To that end, the introduction of benchmark problems and datasets has shown to be immensely successful. Carter et al. (1996) introduced the first benchmark problem for examination timetabling. Since then, several International Timetabling Competitions (ITC), supported by The International Series of Conferences on the Practice and Theory of Automated Timetabling (PATAT), have been incredibly prolific at introducing benchmark problems for educational timetabling. The goal of these competitions has been to promote research attention and 'bridge the gap' (McCollum, 2006) between theory and practice by introducing models that more closely resemble reality (McCollum et al., 2010; Post et al., 2016; Müller et al., 2018b). Table 2.2 shows the most notable benchmark problems found in the literature.

Timetabling category	Benchmark problem	Year	Datasets
High School	Beligiannis	2008	8
	ITC2011	2011	50
University Examination	Carter / Toronto	1996	13
	University of Nottingham	1996	2
	University of Melbourne	2002	2
	ITC2007 - Track 1	2007	8
	UniTime	2013	9
University Course	ITC2002	2002	20
	ITC2007 - Track 2	2007	16
	ITC2007 - Track 3	2007	21
	ITC 2019	2019	30

Table 2.2: Overview of common benchmark problems for educational timetabling.

Section 2.1 discusses some of the practical aspects of educational timetabling. Then the following three sections discuss each of the three main educational timetabling problem categories, focusing on the benchmark problems shown in Table 2.2. Section 2.5 elaborates on student sectioning. A majority of this Ph.D. study has concentrated on the university course timetabling problem posed in the ITC 2019, and Section 2.6 introduces the problem in greater detail.

2.1 Practical Timetabling

Educational institutions must periodically, e.g., annually, once per semester or quarter, solve central timetabling problems. These timetabling tasks are essential, as timetable quality has a significant impact on the previously identified stakeholders (Prida Romero, 1982; Sabin and Winter, 1986). It is not uncommon for timetabling personnel (also called timetablers) to develop all or most of a timetable by hand — a task which takes weeks of tedious and error-prone work (Kingston, 2013). Not only is this approach expensive (in man-hours), but the quality of the timetable likely suffers. Therefore, research into automated or semi-automated methods is vital, but only if they become available in timetabling applications.

In practice, in an attempt at reducing complexity, many educational institutions separate planning problems into distinct phases. For example, a common strategy is to solve the student sectioning problem before solving the timetabling problem. Such separation of the problem makes it easier to solve but usually makes it impossible to find a globally optimal solution (Kingston, 2013). Nevertheless, since the problem considered in each phase is typically difficult to solve on its own, this may be the only practical approach. Furthermore, this justifies research into each particular problem.

Institutions require various planning problems solved at different times of the year. Figure 2.1 shows a typical cycle of major timetabling tasks faced by educational institutions. First comes the student sectioning problem, then the semester timetabling problem, and finally the examination problem. In practice, the cycle also contains many other planning problems, e.g., teacher sectioning and student-teacher conference planning. Some institutions may also combine or change positions of problems in the cycle, like solving the timetabling problem before student sectioning.

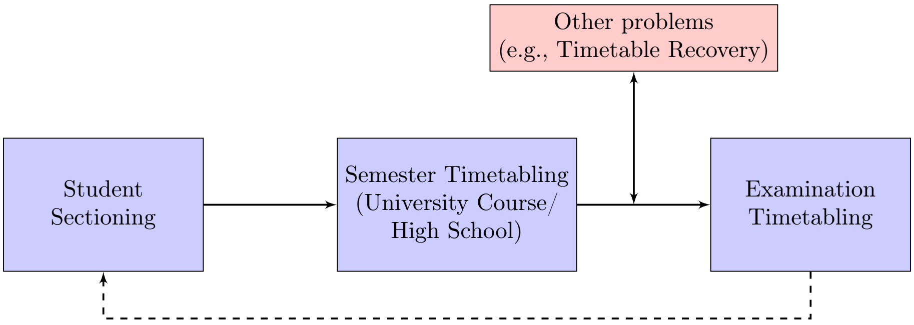

Other problems
(e.g., Timetable Recovery)
Student Sectioning
Semester Timetabling (University Course/ High School)
Examination Timetabling

Figure 2.1: Common cycle of planning problems at educational institutions.

It is utopian to believe that a semester timetable will not experience disruptions. For example, a teacher may become ill or not be available at certain times, or maintenance work may make some rooms unavailable. Thus, another important part of practical educational timetabling is handling disruptions and supporting timetable adjustments (Müller et al., 2005). When changing an already published timetable, it is generally desirable to change it as little as possible to not cause too many inconveniences for affected people. This desire results in the minimum perturbation problem, where the aim is to recover timetable feasibility using as few perturbations (changes) as possible. The minimum perturbation problem has received some research attention; see for example El Sakkout and Wallace (2000), Barták et al. (2004), Müller et al. (2005), Rudová et al. (2011), Fukunaga (2013), and Phillips et al. (2017). However, minimum perturbation solutions often have low quality, and a novel approach suggests that the quality of recovered solutions should play a larger role in minimum perturbation problems (Lindahl et al., 2019). At least, the practical timetabler should be aware of different perturbation alternatives and their associated timetable quality consequences to make an informed decision on the best timetable correction.

As mentioned, each institution faces many different planning problems which must be solved regularly. Additionally, each planning problem has many variations for different institutions. This places a great demand on timetabling models and solvers to be robust. Models should preferably be generic enough to encompass the timetabling problems of different institutions that have wildly varying requirements and constraints. Solvers should similarly be able to solve the resulting varied problems effectively.

As a final note, although each timetabling problem is academically compelling, educational institutions are not particularly interested in proof of optimality, improved bounds on benchmark problems, or similar. They want good (or even just acceptable) timetables for their problems within a reasonable amount of time. Therefore, it is also essential to consider the practical application of educational timetabling research so that it does not become purely theoretical. Such a focus will also help to decrease the gap between practice and theory further.

2.2 High School Timetabling

The secondary school level is for teenagers (approximately 14-18 years old) and includes learning institutions such as high schools. The high school timetabling problem differs for many countries and institutions due to each education systems' varying desired characteristics and requirements. In the context of high school timetabling, the term *class* often denotes a group of students taught a specific subject and a *lesson* to be a specific subject being taught to a class by a teacher. The high school timetabling problem is assigning tuples of classes, teachers, and rooms to times (Pillay, 2010). The timetabling problem does not always contain rooms, either due to pre-assignments or simply because there are plenty of rooms available such that room capacity is not a problem (Post et al., 2012).

As mentioned, the high school timetabling problem can vary a lot for different countries, which may be a contributing factor to the literature focusing on many different timetabling problems. In a 2013 survey on educational timetabling, the authors list 38 articles that focus on 13 country-specific problems (Kristiansen and Stidsen, 2013). Working on different problems makes it difficult to compare to and improve on earlier research, and therefore high school timetabling research has in recent years worked on breaking down this barrier. For example, Pillay (2010) describes the high school timetabling problem in general requirements and constraints that accommodate most problem variations. Some typical hard and soft constraints include:

- Hard constraints:
  - Students, teachers, and classes cannot attend two overlapping events.
  - Classes must be scheduled for the required number of lessons.
  - Lessons must be assigned a time and compatible room when required.
  - Room capacities cannot be exceeded.
  - In terms of lessons/hours per week, a minimum and maximum teacher workload must be observed.
- Soft constraints:
  - Limit idle periods for students and/or teachers[1].
  - Lesson spreading for teachers and classes[1].
  - Lesson and teacher time preferences.

The introduction of the XHSTT format made a great stride in removing the generality barrier, as the format can express unsimplified high school timetabling instances from all over the world (Post et al., 2012; Post et al., 2014). The format results from multiple years of work by a group of researchers motivated to develop a standard format for exchanging datasets for high school timetabling (Post et al., 2012). An XHSTT instance consists of four main parts: times, resources, events, and constraints. Due to the format's modular design, researchers can introduce new constraints without the format requiring general structure changes. Originally, the researchers made efforts to include all constraints found in the literature but, in the end, only implemented the constraints required by contributed instances (Post et al., 2012). Table 2.3 shows the 16 constraint types of the format.

Three attributes define each instance of a constraint: a binary value stating whether the constraint is hard or soft, a non-negative integer weight, and one of the three cost functions shown in Table 2.4. When a solution violates a constraint, the XHSTT format calculates the *deviation* as the size of the violation. The cost of the violation is then calculated by applying the cost function to the deviation and multiplying the result with the constraint weight. The format evaluates the quality of a solution using two values, the infeasibility value and the objective value, which are the sums of violation costs of hard and soft constraints, respectively. In practice, the infeasibility value indicates serious timetable problems, and the objective value shows regular timetable issues.

The website `https://www.utwente.nl/en/eemcs/dmmp/hstt/` (Post, 2021) maintains the XHSTT format, archives of data instances and solutions, provides complete documentation, and more.

[1]Treated as a hard constraint in some variants of the high school timetabling problem.

Constraint name	Description
Assign Resource	Event resource should be assigned a resource
Assign Time	Event should be assigned a time
Split Events	Event should be split into a constrained number of sub-events
Distribute Split Events	Event should be split into sub-events of constrained durations
Prefer Resources	Event resource assignment should come from resource group
Prefer Times	Event time assignment should come from time group
Avoid Split Assignments	Set of event resources should be assigned the same resource
Spread Events	Set of events should be spread evenly through the cycle
Link Events	Set of events should be assigned the same time
Order Events	Set of event pairs should follow each other
Avoid Clashes	Resource's timetable should not have clashes
Avoid Unavailable Times	Resource should not be busy at unavailable times
Limit Idle Times	Resource's timetable should not have idle times
Cluster Busy Times	Resource should be busy on a limited number of days
Limit Busy Times	Resource should be busy a limited number of times each day
Limit Workload	Resource's total workload should be limited

Table 2.3: The constraints of the XHSTT format.

Cost function	Description
Linear	The deviation (no change)
Quadratic	The square of the deviation
Step	1 if the deviation is nonzero, 0 otherwise

Table 2.4: The possible cost functions to apply to XHSTT constraint deviations.

2.2.1 Benchmark Problems

The amount of publicly available datasets for high school timetabling has been somewhat limited.

Beligiannis

The first real-life instances, known as the Beligiannis datasets, were introduced by Beligiannis et al. (2008). These data come from 30 different Greek high schools in the city of Patras. The problem includes typical hard constraints: no double-booking resources and teachers and classes must be assigned precisely the number of events required in periods where the teacher is available. Students are only allowed a given number of periods without teaching, and they must be at the end of the day, i.e., idle periods are not allowed. Additionally, the problem includes the feature of dividing classes (students) into smaller groups that must be taught the same subject simultaneously, for example, to allow for different levels of a subject. It is a hard constraint of the problem that these events occur at the same time.

Soft constraints of the Beligiannis datasets concern idle periods, workload for teachers, and distribution of subject lessons for classes as follows:

- Minimize idle teacher periods.
- Uniformly distribute the total idle periods to all teachers.
- Uniformly distribute individual idle teacher periods on the teacher's available days.
- Teacher workload per day should not differ dramatically among the teacher's working days.
- Classes should not have lessons of the same subject in consecutive hours. Preferably not even on the same day.

ITC2011 / XHSTT

Post et al. (2016) details the Third International Timetabling Competition (ITC2011) and its results. The competition focused on automated high school timetabling and ran from October 2011 to May 2012. All data used the XHSTT format, making it possible to include 35 instances of high school timetabling from 10 different countries. The competition consisted of three distinct rounds with different rules and datasets. In the first round, participants should find the best possible solution on 21 published instances (the XHSTT-2012 archive) with no restrictions. In the second round, the participants were only allowed to use freely available software libraries (disallowing commercial MIP solvers) and 1,000 seconds of execution time on the organizers' computer. The organizers used 18 hidden instances (XHSTT-ITC2011-hidden archive) for scoring. Following the second round, the organizers published the hidden instances for use in the third round. The third and final round, similar to the first round, had no restrictions.

Since the competition used the XHSTT format, the benchmark problem posed by the ITC2011 directly ties to the problem defined by the format. As discussed, the XHSTT-website (Post, 2021) actively maintains the benchmark project and has added new data instances since the competition. Additionally, during the competition, the "Order Events" constraint was not part of the XHSTT format (added December 2012).

2.3 University Examination Timetabling

It is common for university courses to have a final examination at the end of each semester, giving rise to the university examination timetabling problem. The problem can be defined as assigning a set of exams to a limited number of periods and rooms while observing a set of constraints. In this sense, examination timetabling is closely related to high school timetabling and course timetabling, i.e., assigning events (exams/lessons/lectures) to periods and rooms. Schaerf (1999) specifically notes that university course timetabling and university examination timetabling are similar to the extent that it is difficult to make clear distinctions between the two problems. Similarly, McCollum (2006) comments that many pieces of research appear to default to discussing examination timetabling when considering university timetabling in general.

Although the two problems have similarities, there are definitive differences between them, both in terms of problem characteristics and the surrounding practical circumstances (Schaerf, 1999; McCollum, 2006).

Although university examination and course timetabling involve many of the same resources, the examination problem is easier to model (Kingston, 2013). As the examination session generally appears at the end of the semester, student enrollments are ordinarily well known, avoiding the problematic dynamic aspect of course timetabling. Additionally, the room requirements of exams are generally more uniform. Some problems even ignore individual rooms and only consider the total number of seats available. Another benefit of the examination session being placed later in the semester, and the more static nature of the problem, is that timetablers have more time to solve the problem in practice.

University examination timetabling has been the subject of much research. Qu et al. (2009) is a highly regarded survey that thoroughly covers university examination timetabling papers from 1996 to 2009. General double-booking and resource capacity constraints typically constrain the optimization problem. In examination timetabling, as opposed to course timetabling, student conflicts are generally not allowed. Furthermore, it is characteristic for examination timetabling problems to include proximity constraints expressing undesirability of students attending multiple exams close together in time (Kingston, 2013). Proximity constraints spread out conflicting exams and include a measure of "fairness" in the examination timetable, allowing students time to prepare between exams. These constraints can be defined in multiple ways and can be both hard or soft, depending on the problem. Different examination timetabling problems also include other types of soft constraints (Qu et al. (2009) list 11 "most common" soft constraints), but all include proximity constraints.

2.3.1 Benchmark Problems

University examination timetabling has a long history of using benchmark problems and data.

University of Toronto

Carter et al. (1996) introduced the first university examination timetabling benchmark problem known as the University of Toronto or Carter datasets. This set of instances includes real-world examination problems from three Canadian high schools, five Canadian universities, one American university, one British university, and one Saudi Arabian university (Qu et al., 2009). Different variations of the problem and data have been introduced throughout the years, causing some difficulties discerning which specific problem given research has been using. Qu et al. (2009) provides an in-depth discussion on the problem variants, some problems related to duplicate student-exam assignments for some instances, and a mapping of what problem variants specific pieces of research used. Table 2.5 shows an overview of the Toronto benchmark dataset variants (denoted Toronto *a* - *e*). Toronto *a* and *b* were both introduced by Carter et al. (1996). The other variants are extensions of Toronto *b*.

The only hard constraint included in Toronto *a* is that no student double-booking is allowed. Toronto *b* also disallows student double-booking, but also imposes a limit on the number of timeslots available. The limits come from the results of Carter et al. (1996) on Toronto *a*. The objective of Toronto *a* is to minimize the total number of timeslots required for the examination session, i.e., this variant is a graph coloring problem. The objective of Toronto *b* is to minimize the average cost per student using an elaborate proximity constraint. For each student sitting two exams s timeslots apart, a cost w_s is incurred as follows: $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, and $w_5 = 1$. Toronto *b* has received the most research attention of all five variants (Qu et al., 2009).

Variant	Objectives	Modification	Reference
Toronto *a*	Minimize number of timeslots		(Carter et al., 1996)
Toronto *b*	Spread out conflicting exams	Limited timeslots	(Carter et al., 1996)
Toronto *c*	Minimize adjacent conflicting exams on same day	Capacitated (max seat capacity per timeslot)	(Burke et al., 1995)
Toronto *d*	Same as *c* + overnight	Capacitated and varied timeslots per day	(Burke et al., 1998)
Toronto *e*	Same as *c*	Estimated timeslots and capacity per timeslot	(Terashima-Marín et al., 1999)

Table 2.5: The Toronto variants as identified by Qu et al. (2009). Variants *c* - *e* are modified versions of *b*.

University of Nottingham

In addition to introducing Toronto *c*, Burke et al. (1995) introduced the University of Nottingham benchmark set. The problem definition of Toronto *c* and Nottingham is the same, i.e., the objective is to minimize the number of students with consecutive exams on the same day, subject to student double-booking and total seat capacity hard constraints. Burke et al. (1998) extended the problem also to penalize consecutive exams overnight (like Toronto *d*). The two Nottingham variants, with and without counting adjacency overnight, are known as Nottingham *a* and Nottingham *b*.

University of Melbourne

Merlot et al. (2003) introduced the Melbourne benchmark set, including one dataset for each semester of the 2001 academic year at the University of Melbourne. The authors provide a discussion of some of the practical timetabling issues faced by the university. For example, university policy dictates scheduling large exams early in the examination session as a soft constraint. However, university timetablers consider it a hard constraint in practice and prohibit relevant exams from being assigned later timeslots. Another example is that timetablers merge exams required to be held simultaneously into a single exam. This change is possible because the problem considers the total number of seats per timeslot instead of individual rooms. The timetablers use multiple similar "reductions" to convert many constraints into two hard constraints: a total seat capacity limit per timeslot and exam (un)availability for sets of timeslots.

The practical Melbourne examination timetabling included the unique concept of not prohibiting student conflicts. Instead, they opted to prohibit students from having more than two exams per day, using the fact that there are only two timeslots per day. In situations with student conflicts, they resolve the problem using quarantining of relevant students. The authors believed that this practice was inconvenient and unnecessary and enforced the typical, strict no student double-booking constraint.

The objective of the Melbourne problem is the same as for Toronto d and Nottingham, i.e., minimizing the amount of adjacent conflicting exams on the same day and overnight.

ITC2007 - Track 1

Track 1 of the Second International Timetabling Competition (ITC2007) introduced a model of the university examination timetabling problem that is closer to practice, both in terms of data, constraints, and evaluation (McCollum et al., 2007). This model significantly extends the common research models previously worked on (primarily Toronto variants). For example, the ITC2007 model uses explicit rooms defined with a capacity and defines periods with a specific structure and length. The soft constraints of the model are also more closely aligned with how universities typically measure the quality of a solution. In the same spirit, soft constraint weights and parameters are defined in the data (and not in the problem definition), as it is common for universities to experiment with different settings to find satisfactory solutions. Additionally, data-specific weights encourage the development of robust solvers, which should be more usable for different universities.

The ITC2007 model uses three levels of constraint types; required, hard and soft. Required constraints cannot be violated. These constraints dictate that solutions cannot split exams across multiple rooms or periods. Subsequently, the problem evaluates solutions using two values: the number of violated hard constraints denoted "distance to feasibility" and the weighted soft constraint violations. In practice, violated hard constraints would never be allowed, and the practical timetabler would circumvent the problem by introducing new periods, rooms, or similar. Furthermore, McCollum et al. (2007) notes that feasible solutions are generally easily found for the presented problem, and the organizers used this evaluation scheme to be consistent with the other tracks of the competition.

The ITC2007 examination timetabling problem involves assigning each exam to a room and a period. Unlike the previous benchmark problems, exams are assigned to specific rooms, and it is not sufficient to only account for the total number of seats per period. Additionally, exams and periods have varying lengths, and each exam requires an assignment to a sufficiently long period. A unique feature of the problem is that it allows for assigning multiple exams to the same room in the same period, as long as room capacity is respected. The examination model includes typical constraints, e.g., exams must be assigned, no double-booking of students, and exams have period and room preference/unavailabilities. The following text highlights some of the noteworthy constraints.

- Hard constraints:
  - Period precedence: Exam pairs where one exam must occur strictly after the other.
  - Period exam coincidence: Exam pairs where both exams must be assigned the same period
  - Period exclusion: Exam pairs where both exams cannot be assigned the same period
  - Room exclusive: Exams that must have exclusive use of a room, i.e., no other exams can be assigned the same room in the same period.
- Soft constraints:
  - Proximity constraints:
    - Two exams in a row: A penalty applies when a student has two back-to-back exams on the same day.
    - Two exams in a day: A penalty applies when a student has two exams on the same day that are not adjacent.
    - Period spread: The university defines a period spread parameter. Then for each exam, a cost incurs for the number of students who have other exams scheduled afterward within the given parameter.
  - Mixed duration: Penalties apply for room and period combinations that have exams of different duration assigned.
  - Front load: A cost incurs for each large exam (by the number of students) scheduled late in the examination session. The university defines what counts as large, late, and the size of the penalty.

The benchmark problem consists of 12 anonymized data instances from real universities (McCollum et al., 2010). The competition released the instances in three sets of four. The first set was available immediately at the start of the competition, and the second set was published two weeks before the end of the competition. After the competition deadline, the organizers used the final set as hidden verification instances.

UniTime

The UniTime examination sets introduced by Müller (2016) present another real-life university examination timetabling problem. UniTime is a timetabling system, which many universities around the world use (Müller, 2016; Müller et al., 2018a). The benchmark data consists of nine instances from Purdue University from 2008 to 2012. Purdue University is a large American university, and the corresponding examination timetabling problem has about 2,000 exams, 34,000 students, 120,000 student enrollments, 29 periods, and approximately 350 rooms. For comparison, the most extensive set sizes of ITC2007 competition instances are 1,096 exams, 16,439 students, 80 periods, and 49 rooms. The authors argue that the UniTime benchmark problem is especially applicable to larger American universities with extensive exams offered to students across multiple curricula. However, the problem should also be usable for other universities, supported by the fact that the examination problem is implemented in UniTime.

Although the ITC2007 examination track also dealt with "real-world" university timetabling, several differences exist between the benchmark problems. Müller (2016) highlights the following main differences, considering the UniTime problem:

- Student conflicts are allowed but minimized.
- Room sharing is not allowed.
- Exams may be split across multiple rooms (incurring a penalty).
- Rooms have two capacities: examination and regular seating. Regular seating is the number of seats in the room, and examination seating is typically half, such that no students sit next to each other.
- Room and period restrictions/preferences can be set for individual exams. The ITC2007 problem applies such constraints to all exams.
- The problems differ on proximity constraints. UniTime considers direct and back-to-back student conflicts and more than two exams a day. The ITC2007 problem uses back-to-back, more than one exam, and period spread proximity constraints.

Compared to the other benchmark problems, this examination problem is more complex regarding rooms. The problem defines each room with a regular and examination seating capacity, GPS location, period preferences, and restrictions. The other problems do not include any soft constraints concerning rooms (except a global room penalty in ITC2007). The UniTime problem includes six: Same Room, Different Room, Room Penalty, Room Split, Room Split Difference, Room Size, and Room Distance. The Room Split penalizes assigning exams to multiple rooms, and Room Split Distance adds an additional penalty if these rooms are far apart. The Room Size constraint adds a measure of room utilization, applying a penalty for using rooms too big for an exam. The Room Distance constraint is only relevant for some exams and adds a preference that the exam should use the same room as the associated class used during the semester. Additionally, the distance between rooms also has an impact when evaluating back-to-back student conflicts by adding a penalty when rooms are sufficiently distant.

Since real-world universities use the UniTime timetabling suite, and thus the examination problem, it is reasonable that the team behind the system will continuously extend and improve the problem. For example, the team has implemented allowing room sharing of examinations of the same duration, controlled by input data. Additionally, Müller (2016) regards splitting exams into multiple examination periods as an interesting research direction.

Italian Examination Timetabling

Battistutta et al. (2020) introduced a novel examination timetabling problem originating from Italian universities. The Italian Examination Timetabling (IETT) benchmark problem includes 40 real-world instances from seven different departments of six universities. Only recently introduced, the benchmark problem has not yet seen broad adoption but certainly has the potential. The IETT problem is discussed here because a paper included in this thesis (Chapter 10) investigates the application of several techniques on the problem.

This benchmark problem differs from the others in the literature on several aspects:

- The problem contains three exam types: written ($\mathcal{W}$), oral ($\mathcal{O}$), and written + oral ($\mathcal{W}+\mathcal{O}$). For $\mathcal{W}+\mathcal{O}$ exams, the parts should be scheduled at a suitable temporal distance apart.
- A course exam may repeat several times, with a preferred minimum distance between sessions.
- An exam may require a composite room, i.e., a large room resulting from physically combining multiple rooms. Using a composite room concurrently makes all “member” rooms unavailable for other exams.
- The problem is curriculum-based[2]. A curriculum contains both primary courses and secondary courses.
  - Students attend primary courses in the current semester, and conflicts are forbidden.
  - Students attended secondary courses in the previous semester, so some may have to retake exams, and conflicts are discouraged.

Furthermore, the problem contains several proximity constraints that link exams across different groups of exams. Table 2.6 shows an overview of the IETT proximity constraints. As shown in the table, the problem includes hard precedence constraints guaranteeing the order of some event pairs. Additionally, parameters define the preferred minimum/maximum distance for each event pair, and the problem penalizes the size of a given violation linearly.

[2]The following section on university course timetabling (Section 2.4) describes the concept of a curriculum.

Event pair connection	Type	Precedence
Exams of same course	min	Yes
$\mathcal{W} + \mathcal{O}$ parts of same exam	min	Yes
$\mathcal{W} + \mathcal{O}$ parts of same exam	max	Yes
Primary/primary courses of curriculum	min	No
Primary/secondary courses of curriculum	min	No

Table 2.6: Overview of IETT proximity constraints, including the distance direction and the existence of precedence constraints.

2.4 University Course Timetabling

The University Course Timetabling problem consists of assigning lectures of courses to a limited number of times and rooms, subject to hard and soft constraints. The large size and complex structures of universities make the course timetabling problem challenging to solve. It is not uncommon that the complete timetabling problem is divided into smaller problems and solved independently, e.g., for each faculty or department, or that the previous year's timetable is largely re-used (Kristiansen and Stidsen, 2013). Likewise, some solution approaches use multiple stages, which handle different aspects of the problem separately and at different levels (Rudová et al., 2011).

The university course timetabling literature often divides into two main categories: Curriculum-based Course Timetabling and Post Enrollment-based Course Timetabling, usually abbreviated CB-CTT and PE-CTT, respectively.

In curriculum-based timetabling, curricula published by the university predefine conflicts between courses. A curriculum is a grouping of courses that should be taken in combination, for example, to satisfy the degree requirements of a study. A course can be part of multiple curricula, and the timetabling problem consists of scheduling all lectures such that there are no conflicts between courses of the same curriculum.

In post enrollment timetabling, student course enrollments directly define conflicts. The timetabling problem is then to schedule all lectures such that all students can attend their lectures.

Thereby, the main difference between the two problems arises from university structure and time of timetabling. In curriculum-based timetabling, student enrollments are unknown, and the university must develop the timetable using curricula as "lookahead". However, the two approaches are not necessarily incompatible in a practical setting. A university may use a curriculum-based approach to develop a basic timetable, which is later improved using a post enrollment approach once enrollments are known (Kristiansen and Stidsen, 2013).

Although the two problems are different, many constraints are generally the same. Typical constraints are:

- Hard constraints:
  - No double-booking of students, teachers, and rooms.

    - Assigning lectures to rooms satisfying all required features.

- Soft constraints:
    - Lecture room and period preferences.
    - Observing room capacities.
    - Compact timetables for each student or curriculum.
    - Lectures of the same course should use the same room.

2.4.1 Benchmark Problems

University course timetabling benchmark problems originate heavily from international timetabling competitions. Especially the two problems coming from the ITC2007 have received much research attention. The ITC 2019 also involves university course timetabling, and Section 2.6 discusses this benchmark problem.

ITC2002

The First International Timetabling Competition (ITC2002) presented a basic form of the post enrollment university course timetabling problem (Rudová et al., 2011). The organizers designed the problem to be a reduced version of a typical university course timetabling problem (Paechter et al., 2002). The problem consists of a set of events (lectures) to schedule in 45 timeslots, a set of rooms with capacities and different features, and students with enrollments. The hard constraints of the problem include student and room double-booking and that events must be assigned rooms with sufficient capacity and requested features. Soft constraints focus on individual student timetables, applying a penalty for each student that has: an event scheduled in the last timeslot of a day, precisely one event scheduled in a day, and more than two events scheduled consecutively.

The competition data consists of 20 computer-generated instances designed to have varying difficulties. However, for each instance, there exists at least one perfect solution, i.e., a solution with an objective value of 0. The competition released the instances in two sets, the initial ten at the start of the competition and the final set two weeks prior to the deadline.

As is the case with all the international timetabling competitions, the goal of the ITC2002 was to attract research attention by establishing a specific benchmark problem with associated data to allow for comparisons between different solution methods (Bettinelli et al., 2015). To that end, the competition was successful, resulting in several publications. Kostuch (2003), Cordeau et al. (2003), Bykov (2003), and Di Gaspero and Schaerf (2003) describe the top four performing methods (in order). Kostuch (2004) presents an improved version of the winning method.

ITC2007 - Track 2

Track 2 of the ITC2007 presented an extended version of the post enrollment university course timetabling problem from the ITC2002 (Lewis et al., 2007).

The extension consists of two additional hard constraint types: allowing for events to have predefined available timeslots and to define event precedence, enforcing specific events to be scheduled strictly before others. The extended problem definition includes no changes to the soft constraints or objective function. However, the competition used a different evaluation scheme compared to the ITC2002. The competition organizers noted that the hard constraints of the ITC2002 instances were generally easy to satisfy and that inclusion of the additional constraints made it unrealistic to expect all algorithms to find feasible timetables within the competition time limits. Therefore, following the other tracks of the ITC2007, solutions were evaluated using both distance to feasibility and an objective function. However, the competition did not allow solutions with violated hard constraints but instead opted to allow unassigned events. Thus, a competitor could submit a timetable with violated hard constraints by removing the offending events. Consequently, the problem calculated the distance to feasibility as the number of students attending all unassigned events.

The organizers included the additional constraints and used the described evaluation scheme to move the benchmark problem closer to reality. They argue that in real-world timetabling, the timetabler might also use the distance to feasibility followed by the objective value. In a practical setting, the timetabler wants to satisfy as many people as possible, which the distance to feasibility measures by not considering the number of unassigned events but the number of affected students.

Although this benchmark problem resembles a real-world post enrollment university timetabling problem more closely than the ITC2002 problem, it intentionally excludes some real-world constraints and features. Chiefly this is a consequence of the competition setting, as it was necessary not to overwhelm participants and maintain a degree of generality. Lewis et al. (2007) highlights several potential features and constraints to include. These considerations include inter-site travel times, scheduling lunch breaks, the relative timing of events, allowing events of different duration, and including free days and lecturer preferences objectives. Additionally, they reflect on room-related features such as events without rooms, room-period availability, room utilization objectives, and the use of rooms that can partition into smaller rooms.

The benchmark data consists of 21 generated data instances. Similar to the ITC2002, the data instances are of varying difficulty, but each has at least one perfect solution. The competition published the 21 instances in three sets of seven. The first set was made available at the beginning of the competition, the second set two weeks before the deadline, and the final set was used as a hidden evaluation set and released following the competition.

ITC2007 - Track 3

Track 3 of the ITC2007 presented a curriculum-based course timetabling problem described in detail by Di Gaspero et al. (2007). As the problem is curriculum-based, it does not include students and avoids conflicts by using curricula. The problem and data originate from the University of Udine. Like the other tracks of the ITC2007, the track uses a simplified problem and data to maintain generality and not overburden competition participants. However, the organizers note that the formulation still applies to many Italian and international universities.

The problem consists of periods (with associated days), courses with a fixed number of lectures, students and a teacher, rooms with a capacity, and finally, some curricula. The problem is to schedule all events while avoiding all room, teacher, and student (from curriculum) conflicts. The only other hard constraint of the problem is event-period availability, primarily determined by teacher availability. To model cases where days have a different number of periods, the problem uses event-period availability and makes all courses unavailable in specific periods. The soft constraints relate to event-room assignments, spreading and room usage of lectures of the same course, and compactness of curricula, as follows:

- Room capacity (weight 1): Lectures should be assigned rooms with sufficient capacity. A penalty applies for each student above the capacity.
- Room stability (weight 1): Lectures of the same course prefer to use the same room. A penalty is applied for each additional distinct room used by the lectures.
- Minimum working days (weight 5): The timetable should spread lectures of the same course into a given minimum number of days. A cost incurs for each day below the minimum.
- Curriculum compactness (weight 2): Individual student timetables should be compact. A penalty applies whenever a single lecture is not adjacent to any other lecture of the same curriculum on the same day. Specifically, the problem allows for idle periods but forbids a single lecture from being completely isolated and surrounded by idle periods.

Like track 2 of the ITC2007, this benchmark set consists of 21 data instances released in three sets of seven. All instances are real-world data gathered from the University of Udine. Following the competition, some additional sets of instances have been made publicly available by different contributors. These are both suitable for the ITC2007 CB-CTT problem formulation, extended versions of it, and sometimes used together with the competition instances for benchmarking (Bonutti et al., 2012). The sets come from Udine University, other Italian universities, and the University of Erlangen. The Erlangen instances are significantly larger than those used in the competition and include a very different structure compared to other real-world instances (Bellio et al., 2016). The organizers also provide an instance generator, which researchers can use to train algorithms before validating and benchmarking on real instances.

2.5 Student Sectioning

Courses with a large student enrollment sometimes need to be partitioned into sections, typically due to room capacities or resource limitations. Each section offers a separate copy of the course and must be assigned an individual time, teacher(s), and room. The presence of course sections introduces the need for solving the student sectioning problem, that is, assigning students to exactly one section of each of their required courses. The student sectioning problem has not received much research attention and often falls outside the classical educational timetabling problem categorization (Müller and Murray, 2010). In the other problems, events demand times, teachers, and rooms, but in student sectioning, students demand courses (events) (Kingston, 2014). Student sectioning arises at both universities and high schools, but the high school problem is generally smaller and less well studied (Kristiansen and Stidsen, 2013). Some examples of high school student sectioning are Haan et al. (2007), Kristiansen et al. (2011), Kristiansen and Stidsen (2016), and Hoshino and Fabris (2020).

How and when to solve the student sectioning problem varies greatly, both in practice and the literature. Student sectioning may be solved both before, during, and after general timetable development. When solving student sectioning before the timetable, the goal is to find a sectioning that supports subsequent timetabling by avoiding potential student conflicts. When solving student sectioning concurrently or after solving the timetabling problem, the aim is to avoid definite individual student conflicts. The literature is split regarding the order, but most solve student sectioning after developing the timetable (Kingston, 2014; Schindl, 2019).

However, some cases consider student sectioning in multiple phases. One notable work by Müller and Murray (2010) presents a comprehensive timetabling approach that focuses on student requests and sectioning, both during and after timetable development. Their algorithm works in three separate phases. First, the algorithm develops the timetable while minimizing potential student conflicts. Then the algorithm assigns all enrolled students to a section while considering projected additional enrollments. Students may change their enrollments online in the final phase, but the algorithm remains attentive to possible future enrollments.

Many consider student sectioning to be a subproblem or a separate phase of timetabling (Müller and Murray, 2010). Many later works disregard student sectioning completely, decreasing problem complexity at the cost of being less relevant to real-world problems (Rudová et al., 2011). This practice is understandable considering how the two problems are different from each other. In addition, the timetabling problem is already difficult to solve, and adding student sectioning only increases the complexity. In fact, most student sectioning problems are $\mathcal{NP}$-complete. Dostert et al. (2016) presents that a fundamental student sectioning problem, which has a fixed timetable and all students must attend all courses, has polynomial complexity. However, they also show that adding just one of the following typical constraints/features makes the problem $\mathcal{NP}$-complete.

- Students enroll in a subset of courses (as opposed to all).
- Students have individual timeslot restrictions.
- Sections can have multiple events (that cannot clash) to assign timeslots, i.e., the same section may occupy multiple timeslots.

Student sectioning presents both an essential and challenging timetabling problem, that unfortunately, has received little research attention. Consequently, the student sectioning problem does not have any broadly adopted benchmark problems. However, the data used by Müller and Murray (2010) is publicly available on the UniTime website for use as a benchmark set.

2.6 International Timetabling Competition 2019

The ITC 2019 presents a novel benchmark problem combining university course timetabling with student sectioning. The problem solves student sectioning and assigns courses to times and rooms simultaneously in one phase. The ITC 2019 problem poses a very powerful and sufficiently generic problem definition that can be used in many universities with very different structures worldwide. Consequently, the ITC 2019 problem has the potential to be significant both as a benchmark problem and for solving practical problems. This section describes the ITC 2019 problem in greater detail since much of this Ph.D. study has focused on it. Müller et al. (2018b) provides a complete description of the competition and the problem.

The course timetabling problem originates from UniTime, which several universities use in practice. The fact that the competition instances come from 10 different universities in eight countries on five continents further demonstrates the generality of the problem definition. The timetabling problem is post enrollment-based, and curriculum-based problem instances are handled by transforming curricula into course demands. In order to make the problem more attractive in a competition setting, some less critical aspects of the real-world problem have been removed or simplified without reducing complexity. This section highlights the most significant changes after introducing the problem.

The problem consists of four main components: rooms, classes, distribution constraints, and students. The goal is to assign classes to rooms and times while also sectioning students into classes while observing all hard constraints and minimizing the total soft constraint and preference penalty.

Students enroll in courses that have a potentially complex structure. Each course contains one or more configurations, which all consist of one or more subparts. Each student must attend exactly one class from each subpart of a single configuration for each of their courses. A strict parent-child relationship may exist between classes of different subparts of a configuration, such that students must attend the parent class if attending the child class. Additionally, each class has a strict capacity limit to observe. This complex hierarchical course structure allows for modeling many different university arrangements and for a single course to allow for very different configurations. For example, a course may have one configuration with lectures and recitation and another configuration with lectures, recitation, and laboratory work.

All competition instances consist of five-minute timeslots, but the format allows for other timeslot lengths. Classes are not assigned to a single timeslot but instead a time pattern, defined by a starting time (timeslot), the duration (in timeslots), and the days and weeks where meetings occur. This format allows a class to meet several times a week during specific weeks but always at the same starting time and for the same duration. Additionally, the short timeslot length allows for diversified and varying meeting times. The problem instance defines each class with all its possible time patterns, each with a non-negative penalty stating the class-time preference.

The problem defines rooms with a capacity and possibly a list of times when the room is unavailable. Symmetric travel times between room pairs are also defined. If a class requires a room, the format includes a list of rooms that comply with all the class's feature requirements. Similar to times, the format defines each class-room pair with a non-negative penalty.

As mentioned earlier, the problem defines students with their course enrollments and requires the sectioning of each student, such that the student attends one class from each subpart of a single configuration for all their courses. In this problem, student conflicts (double-booking) are allowed but penalized. A student conflict occurs when a student is enrolled in two classes that overlap in time or when the two classes are scheduled in rooms sufficiently far apart. Specifically, a student conflict occurs when the travel time between the two rooms is greater than the number of timeslots between the two classes.

The problem contains several distribution constraints which place restrictions or imposes preferences on timetable characteristics. Table 2.7 shows the 19 distribution constraint types. Some constraint types are evaluated in a pairwise fashion, allowing, for example, for evaluating a constraint placed between four classes using the resulting six class pairs. The problem defines constraints as hard or soft with a given non-negative penalty. The following text highlights a few distribution constraints.

Constraint	Opposite	Pairwise
`SameStart`		✓
`SameTime`	`DifferentTime`	✓
`SameDay`	`DifferentDay`	✓
`SameWeeks`	`DifferentWeeks`	✓
`SameRoom`	`DifferentRoom`	✓
`Overlap`	`NotOverlap`	✓
`SameAttendees`		✓
`Precedence`		✓
`WorkDay(S)`		✓
`MinGap(G)`		✓
`MaxDays(D)`		–
`MaxDayLoad(S)`		–
`MaxBreaks(R,S)`		–
`MaxBlock(M,S)`		–

Table 2.7: Distribution constraints of the ITC 2019 problem (Müller et al., 2018b).

The `SameAttendees` constraint indicates that it should be possible for the same people to attend the given classes. The same rules apply as for student conflicts: the classes should not overlap in time, and there should be sufficient time between classes to allow for travel time. This constraint is essential for ensuring that instructors can attend all their events and to enforce the valid distribution of parent-child classes. The `Precedence` constraint enforces strict ordering of classes and can be used to, for example, ensure that a required lecture takes place before a seminar.

The last four constraint types of Table 2.7 require all classes of the constraint for evaluation. The `MaxDays(D)` constraint involves spreading classes over `D` days, regardless of weeks, i.e., the timetable should have the classes on at most `D` days of the week. The `MaxDayLoad(S)` spreads the classes over days, such that the classes use no more than `S` timeslots on any day. The `MaxBreaks(R,S)` constraint imposes a limit on the number of breaks of the given classes. The problem defines a break to be idle time between classes of more than `S` timeslots and allows no more than `R` breaks per day. The `MaxBlock(M,S)` constraint disallows more than `M` blocks of consecutive classes per day. The `S` parameter controls when gaps are sufficiently large to introduce a new block. The latter three constraint types require integer division when evaluating the soft constraint penalty.

Contrary to some of the other competitions, the ITC 2019 does not include a distance to feasibility value and allows only feasible solutions for submission. The weighted sum of time assignments, room assignments, student conflicts, and distribution constrain penalties determines the quality of a solution. Each of the four optimization criteria has an associated weight parameter, which allows the institution to prioritize and control its timetabling problem.

The competition separated the data sets into three sets of 10 instances, released at different times during the competition as hinted by the set names: early, middle, and late. Competitors seek to provide the best possible solution for each data instance. To determine the final ordering of competitors, the organizers use a scoring system similar to the F1 championship, where instances released later are worth more points. An online validator, which can also validate custom data instances, is available on the competition website (`https://www.itc2019.org`).

The data instances come from a variety of universities with very different characteristics. One notable difference is the size of the problems. Some instances only consider a single school or faculty, while others involved complete universities. Student course demands also vary from instance to instance. For example, some consider pre-enrollments, some curriculum-based enrollments, and some do not even include students. For instances without student course demands, distribution constraints like `SameAttendees` or `NotOverlap` enforce the required distribution of classes. Another important difference is the complexity of courses. Some universities have straightforward course structures, perhaps consisting of single lectures, while others require many diverse configurations for courses. Also, some universities use non-overlapping timeslots while others allow for irregular meeting times.

The fact that the competition format can contain all the mentioned problem characteristics is a testament to its generality. However, the organizers made some simplifications and transformations of the real-life data to fit the competition problems, but primarily for rarely used features. For example, some distribution constraints have been completely removed or approximated using other included distribution constraints. Some room features were also simplified. The organizers combined characteristics of complex room availability and specific class penalties to create the list of rooms with penalties for each class. The organizers also exclude a room utilization penalty, which penalizes classes assigned to rooms with much excess capacity. In practice, a few problems include additional features for student sectioning like specific student-class reservations, forbidding or using different weights for specific student conflicts, and trying to keep some students grouped. More aspects have been left out or approximated in the competition format, but since they are rare, this should not make a significant problem for the real-world applicability of competition solution approaches.

2.6.1 Competition Results

The competition concluded with an award ceremony, which included an overview of the competition, presentations of finalists, and the final ordering of the competitors. Table 2.8 shows an overview of the five finalists, their solvers, and their competition score. MIP-based approaches won both first and second place. Our team (Holm et al.) won first place using a parallelized matheuristic developed as part of this Ph.D. study and described in detail in Chapter 9. This solver found the best solution on 29 out of the 30 scoring instances. Rappos et al. implemented a solver consisting of two stages. First, they find a feasible solution by solving a sequence of relaxed MIP models using variable fixing and gradually adding violated constraints. The second stage improves the feasible solution by iteratively fixing variables and solving the model. Gashi and Sylejmani implemented a simulated annealing algorithm with an adaptive evaluation function, in which they allowed infeasible solutions. They also included a specialized hill climber to escape persistent infeasibilities. Er-rhaimini implemented a forest growth algorithm using random greedy construction heuristics and solution refinement. Finally, Lemos et al. implemented a solver that used MaxSAT to develop a feasible timetable and a local search heuristic to improve student conflicts.

The competition website (`https://www.itc2019.org`) includes the award ceremony presentations and summary articles detailing each competitor's solver. Additionally, it provides a complete overview of the results, including the cost of the best solution found by each team and the associated score. Although the competition has concluded, the organizers will maintain the website. Academics and practitioners are encouraged to continuously upload solutions from their solvers to keep track of the state-of-the-art and compare solvers.

Following the competition, there have been some submissions to the website, such as competitors uploading improving solutions. However, most significantly, Tomáš Müller, one of the competition organizers, has applied the UniTime solver (with minor adjustments) to the ITC 2019 problem and uploaded the results (Müller, 2020). The solver (described in more detail in

Place	Team	Summary Paper	Method	Score
1	Dennis S. Holm, Rasmus Ø. Mikkelsen, Matias Sørensen, Thomas J. R. Stidsen (MaCom / Technical University of Denmark, Denmark)	(Holm et al., 2019)	Matheuristic	489
2	Efstratios Rappos, Eric Thiémard, Stephan Robert, Jean-François Hêche (HEIG-VD, Switzerland)	(Rappos et al., 2019)	Matheuristic	322
3	Edon Gashi, Kadri Sylejmani (University of Prishtina, Kosovo)	(Gashi and Sylejmani, 2019)	Simulated Annealing	273
4	Karim Er-rhaimini (Ministère de l'éducation nationale, France)	(Er-rhaimini, 2019)	Forest Growth	252
5	Alexandre Lemos, Pedro T. Monteiro, Inês Lynce (INESC-ID / IST, Universidade de Lisboa, Portugal)	(Lemos et al., 2019)	MaxSAT	79

Table 2.8: Overview of the five finalists.

Section 2.7) won two tracks of the ITC2007 and placed third in the last track. Table 2.9 shows post-competition standing as of July 2021, including the number of best solutions and scores for each competitor. For one Early instance, multiple competitors found the best solution. The UniTime solver has achieved second place and found the best-known solution to 8 instances and a total score of 382. Holm et al. have submitted 23 best-known solutions and have a total score of 449.

		Best solutions			Score			
Place	Competitor	Early	Middle	Late	Early	Middle	Late	Total
1	Holm et al.	10	6	7	97	126	226	449
2	Müller	1	4	3	64	123	195	382
3	Rappos et al.	1	0	0	58	61	123	242
4	Gashi and Sylejmani	0	0	0	20	55	100	175
5	Er-rhaimini	0	0	0	18	45	96	159

Table 2.9: Overview of best solutions and scores as of July 2021.

After the competition, our team (Holm et al.) has improved our solutions on 14 instances in total. Besides uploading improving solutions to the competition website, we also plan to keep an updated overview on `https://dsumsoftware.com/itc2019/`, in which we also show our best-known bounds and gaps for each instance. Additionally, we host competition instances that we have reduced using several techniques, some of which are discussed in Chapter 7.

2.7 Successful Methodologies

This section highlights some recently successful methodologies within educational timetabling using the surveys shown in Table 2.10. The table shows the number of references belonging to different method categories, using, where possible, the author's categorization. Furthermore, many methods are composed of multiple techniques (hybridization) but are classified based on the "main" method. Most of the surveys specifically note that hybridization has become increasingly popular and effective. Hybrid approaches benefit from combining the strengths of multiple methods, often resulting in better performance and application for a wider range of problems. In the same vein, hyper-heuristics have also become increasingly popular. Hyper-heuristics combine multiple methods and use some heuristic for dynamically choosing heuristics during the search. Thus, hyper-heuristics focus on developing a general approach applicable to multiple problems instead of a specialized and problem-specific approach. A recent review on hyper-heuristics for educational timetabling concludes that hyper-heuristics are effective for these problems and have great potential for advancing the field, especially in the direction of generalized solutions for educational timetabling as a whole (Pillay, 2016).

Qu et al. (2009) is the essential survey on educational examination timetabling, providing an overview of benchmark problems, earlier surveys, and relevant literature from 1996 to 2009. The authors argue that, in recent years, metaheuristics have received the most attention, a significant portion of which focus on population-based techniques. They note that hybrid approaches generally have better performance than "pure" algorithms and that future research should investigate how to integrate different techniques successfully. Additionally, they highlight that hyper-heuristics presents another important research direction regarding theoretical study and practical applications.

Pillay (2010) provides an overview of high school timetabling, including problem components, available data sets, and recent research. The survey predates the completion of the XHSTT format, and the discussed methods primarily concern not publicly available data instances. The author notes that a majority of methods employ a construction phase that schedules difficult events first. Most use "most constrained" as the measure of difficulty, and the author calls for investigation of other evaluation heuristics. Additionally, the author argues for more research into the application of hyper-heuristics on high school timetabling problems.

Kristiansen and Stidsen (2013) present an excellent introduction to educational timetabling. The authors discuss practical aspects of planning problems in real-life educational institutions and provide an introduction to the three main educational timetabling problems and student sectioning. For each problem, they discuss recent literature, focusing on solutions that are implemented or tested on real-life data. The authors conclude that the last decade has seen an increasing amount of successful literature using hybrid methods. However, they also call for more research into exact methods to produce better lower bounds.

Bettinelli et al. (2015) present an overview of curriculum-based course timetabling using the benchmark problem of the ITC2007. They focus on mathematical/exact models for lower bounds and the most effective heuristics proposed in the literature, including the competition's five finalists. The authors conclude that metaheuristic and constraint satisfaction-based algorithms generally achieve better results than mathematical model-based algorithms. Especially hybrid methods are often very effective.

Tan et al. (2021) is a very recent survey focusing on the state-of-the-art optimization techniques for high school timetabling. The survey presents comparative studies considering both methodology type and benchmark problem and shows recent solution methods in chronological order. They show that most metaheuristics are non-population-based, and variable neighborhood and adaptive large neighborhood search algorithms are popular. However, the authors note a popularity shift from local search methods to mathematical optimization methods, and integer programming-based methods present the current state-of-the-art for three prevalent benchmark problems. Even so, more than half of the investigated methods are either a hybrid metaheuristic or hyper-heuristic.

It seems that hybrid methods have become increasingly popular and successful solution strategies for educational timetabling problems. Both the ITC2011 and ITC2007 are testaments to the success of hybrid methods. Team Goal won first place in the ITC2011 using hybridization of simulated annealing and iterated local search (Fonseca et al., 2012). Later they extended their solver with a stagnation-free Late-Acceptance Hill Climbing algorithm to find the best results for some XHSTT datasets (Fonseca et al., 2016). Müller (2008) presents the hybrid approach that won the examination and curriculum-based course timetabling tracks and placed fifth in the post enrollment course timetabling track. The method consists of three or four stages. The first stage constructs an initial solution using an iterative forward search. The second stage uses a hill climbing algorithm with multiple neighborhoods until it finds a local optimum. The third stage uses great deluge to escape the local optimum and widen the search. The optional fourth stage uses simulated annealing to continue the search. The excellent performance of this hybrid approach in all three competition tracks demonstrates the power of hybridization for a generalized solution approach, especially with regards to practical applications where tuning to specific problems or problem instances may be undesirable.

Paper	Area	Method category	References	Timespan
(Qu et al., 2009)	Examination	Graph coloring heuristics	4	1999-2004
		Constraint-based	5	1998-2006
		Tabu search	5	2001-2002
		Simulated annealing	5	1998-2004
		Local search (not Tabu or SA)	5	2001-2008
		Evolutionary algorithms	13	1996-2007
		Ant / artificial immune algorithms	5	2004-2007
		Multi-criteria	5	1995-2006
		Hyper-heuristics	12	1999-2007
		Decomposition	3	1999-2007
(Pillay, 2010)	High School	Simulated annealing	2	1991-2006
		Evolutionary algorithms	7	1991-2008
		Tabu search	3	2005-2008
		Integer programming	3	1997-2009
		Constraint programming	1	2003
		GRASP	1	2010
		Tiling	2	2004-2006
		Hybrid	3	1991-2007
(Kristiansen and Stidsen, 2013)	Course	Swarm Intelligence algorithms	9	2003-2013
		Evolutionary algorithms	3	2007-2008
		Local Search (Tabu, SA, GRASP)	8	2003-2001
		Graph coloring	2	2007-2013
		Integer Programming	11	2004-2013
	High School	Evolutionary algorithms	11	1999-2012
		Local Search (Tabu, SA, GRASP)	20	1998-2012
		Graph coloring	1	2008
		Integer Programming	11	2003-2014
		Tilling	1	2005
		Hyper-heuristic	2	2012-2012
	Examination	Swarm Intelligence	5	2005-2011
		Evolutionary algorithms	2	2002-2011
		Local Search (Tabu, SA, GRASP)	11	2003-2012
		Graph coloring	6	2005-2014
		Integer Programming	2	2007-2010
		Hyper-heuristic	10	2003-2013
(Bettinelli et al., 2015)	Course (curriculum)	Exact methods	8	2010-2014
		Integer Programming-based	2	2010-2012
		Tabu search	4	2008-2011
		Simulated annealing	4	2012-2014
		Hybrid	5	2009-2014
(Tan et al., 2021)	High School	Integer Programming	8	2014-2020
		Adaptive Large Neighborhood Search	2	2012-2012
		Variable Neighborhood Search	3	2013-2019
		Particle Swarm Optimization	1	2012
		Cyclic transfer algorithm	1	2012
		Constraint Programming	1	2018
		Graph Coloring	1	2014
		Parallel search	1	2018
		Tabu search	1	2010
		Simulated annealing	2	2010-2020
		Evolutionary algorithms	3	2012-2016
		Matheuristics	3	2014-2020
		Hyper-heuristics	4	2015-2017
		Hybrid	7	2015-217

Table 2.10: Overview of some recent surveys within educational timetabling.

Chapter 3

Overview of Results

This chapter describes the results of this Ph.D. study. Section 3.1 provides an overview of scientific papers produced during the study. Section 3.2 discusses the scientific contributions of the thesis. As this Ph.D. study falls under the Industrial Ph.D. Program, it includes Section 3.3 to discuss the practical applications of the work.

3.1 Papers

This thesis is composed of five scientific papers submitted to peer-reviewed journals and one published technical report. This section provides an overview of each work. Part II includes all works in full length.

Chapter 5: Educational timetabling: Optimizing quality of perturbations

Submitted to Annals of Operations Research

This paper proposes a novel MIP-based approach for providing perturbation suggestions for a given high school timetable. We apply the approach to the high school timetabling problem in Denmark but argue that the method is adaptable for other problem variants. We show how to adjust the MIP model of the problem to force timetable perturbations and to find perturbation suggestions that both prefer positive changes for user-defined timetable entities (students, teacher, and classes) and make sense to the user. We introduce a model approximation as the full MIP model becomes quite comprehensive. We run computational tests on real-world datasets from the Danish student administration system Lectio. The tests show that the full model, compared to the approximation, results in high-quality suggestions, but by less than 1% in average relative gap. Conversely, when using the approximated model, we can produce a greater number of suggestions more rapidly, which is essential since we implement the method in an interactive environment.

Chapter 6: A MIP Formulation of the International Timetabling Competition 2019 Problem

Published as a Technical Report

In this technical report, we detail a MIP model of the ITC 2019 problem. We believe this to be the first complete MIP model developed for the ITC 2019 problem, and we give a complete description of all variables, objectives, and constraints, including the more difficult constraints (last four constraints of Table 2.7) and identified special cases. We also present computational results for all instances except one which was too memory demanding. We show the total number of constraints and variables of the MIP model for each instance, the best solution and bound found by the MIP solver after 1 and 24 hours, and the number of constraints and variables removed by presolve. Only for 10 out of the 29 instances do we have a solution after 24 hours, showcasing that we cannot use the full MIP model as a standalone approach for solving the ITC 2019 problem. However, the approach is not without merit, as it solves three instances to proven optimality and one with an optimality gap of 3.06%.

Chapter 7: A Graph-Based MIP Formulation of the International Timetabling Competition 2019

Submitted to Journal of Scheduling

This paper improves upon the MIP model introduced in the technical report and introduces some preprocessing of instance data. We use different conflict graphs and graph structures to reduce the number of variables and constraints required to model the problem. For example, we use clique covers, star covers, and odd cycles to reduce and tighten the formulation regarding hard conflicts. Additionally, we use a special star cover to reduce the number of constraints required to model soft conflicts. To preprocess the data, we introduce a reduction algorithm that removes redundancies and dominated distribution constraints from the raw instance data. We also describe other analytical reductions such as additional variable reduction by cliques, fixed vertices, inevitable or impossible student conflicts, and more.

We evaluate the reductions and compare the improved graph-based MIP with the 'basic' MIP through computational tests. On average, the preprocessing reduction algorithm can reduce the number of distribution constraints of an instance by 13.67%. Comparing the two MIP models, we achieve an average constraint and variable reduction of 77.18% and 42.96%, respectively. We also compare both models after presolve, for which the graph-based MIP shows a reduction of 63.45% and 10.43% for constraints and variables, respectively. Additionally, the presolve stage requires, on average, 20.5 minutes less. The improved model formulation also results in better performance when solved using a MIP solver. After 24 hours, the MIP solver has found 12 solutions and 27 lower bounds. Comparatively, the basic MIP finds ten solutions and 22 lower bounds in the same amount of time. Not only does the graph-based MIP produce more solutions and bounds within the same time, but all are also generally of better quality.

Chapter 8: A Fix-and-Optimize Matheuristic for the International Timetabling Competition 2019

Submitted to Journal of Scheduling

This paper describes a fix-and-optimize matheuristic and an initial solution constructive matheuristic (called 2SCA) implemented for the ITC 2019 problem. Both methods use the graph-based MIP detailed in Chapter 7. The constructive heuristic uses a relaxed model that only includes hard constraints, disregards student sectioning, and only has the number of unassigned classes in the objective function. We solve the relaxed MIP model to optimality to get a feasible albeit likely poor course timetable. Then we fix all course assignments, add student sectioning to the model and solve the model to get an initial solution. For the fix-and-optimize matheuristic, we describe three distinct heuristics used for choosing course classes that we allow to change assignments during an iteration. Additionally, we detail how we control the neighborhood size and subproblem complexity dynamically throughout the search.

We tested the methods on the 29 competition instances, as one was too memory intensive. We compare the best solutions found within 24 hours using fix-and-optimize with different neighborhoods and directly solving the full MIP model. We warm-start all methods using the solutions obtained by 2SCA. The fix-and-optimize searches generally proved better at finding solutions, with varying success depending on the neighborhood. Only on three instances did solving the MIP model yield better results than a fix-and-optimize search. Additionally, the MIP model did not produce any solutions on ten instances, while all fix-and-optimize searches produced solutions for all instances.

Chapter 9: A Parallelized Matheuristic for the International Timetabling Competition 2019

Submitted to Journal of Scheduling

This paper presents the complete solver that we used to win the ITC 2019. We use the graph-based MIP model and methods described in the other papers, plus some additions, in a parallel framework. The main improvement comes from introducing solution sharing between several fix-and-optimize searches and a MIP solver solving the full MIP model. Solution sharing improves the best found solution by an average of 10.42%, given a 24-hour time limit. We also introduced a special setup for instances with an excessive number of students, which decouples timetabling development and student sectioning. Only by using these methods could we find feasible solutions for an instance too large for the other methods. As a final substantial addition, we introduced a diversification scheme tuned to a 10-day runtime (similar to the 'final' part of the competition). This diversification scheme did find some success but relied too much on luck — an issue we want to address in future research. However, the complete solver proved very competitive and found the best competition solution on 29 out of 30 instances. Using the MIP model, we also produce lower bounds and have proven optimality for five solutions.

Chapter 10: Comparing Exact and Metaheuristic Methods for a Real-World Examination Timetabling Problem

Submitted to Journal of Heuristics

This paper investigates different solution techniques for solving the real-world Italian examination timetabling problem (IETT) described by Battistutta et al. (2020). The problem comes from Italian universities, and uses real-world data instances. We investigate a portfolio of exact and metaheuristic solution approaches. For the exact methods, we developed two MIP models (one encoded natively and the other in the MiniZinc modeling language (see Nethercote et al., 2007)) and a Constraint Programming (CP) model. The CP model presents an improved version of the CP model proposed by Battistutta et al. (2020). We encoded the model in MiniZinc and used it in a heuristic solution approach using Large Neighborhood Search. We also developed a two-stage decomposition using the natively encoded MIP model. The decomposition solves the problem without rooms in the first stage and adds rooms in the second stage. The first stage includes constraints to make infeasibilities in the second stage highly unlikely. Finally, we also presented a simulated annealing approach which is an extension of the one described by Battistutta et al. (2020). In total, we investigated three independent optimization paradigms (CP, MIP, and simulated annealing) and five solution approaches (two independent MIP models, MIP decomposition, CP-based local search, and simulated annealing).

The simulated annealing approach yielded the best results, providing the best average solution for all instances except two. The methods using MiniZinc generally have the worst performance, which is not unexpected, as MiniZinc is a general-purpose, high-level language. Comparatively, the natively encoded MIP model and the two-stage decomposition have better performance. Especially the two-stage approach does well, generally obtaining the best average solution quality of all methods disregarding simulated annealing. Finally, we present the first lower bounds for this problem in the literature. We prove that we have solved 11 of 40 instances to optimality, with ten perfect solutions with a cost of 0.

3.2 Scientific Contributions

This section lists the scientific contributions of this thesis.

- A novel MIP-based approach for providing practical timetabling decision support in the form of timetable perturbation suggestions. The technique is demonstrated on the Danish high school timetabling problem as defined by the Danish student administration system Lectio. The algorithm is implemented in the timetabling suite of Lectio and is available to practical timetablers.

- The current state-of-the-art solver for the university course timetabling benchmark problem presented by the ITC 2019. The solvers is a parallelized matheuristic method that includes the following contributions related to the ITC 2019 problem.

  - A complete MIP model — the first exact method in the literature. The model is analyzed and improved using numerous graph techniques on conflict graphs. The MIP model has yielded optimality proofs for five benchmark instances and the first lower bounds for the rest.
  - Preprocessing technique to tighten the raw data.
  - A fix-and-optimize matheuristic including dynamic subproblem complexity control.
  - Two MIP-based constructive heuristics.
  - A collaboration framework for the fix-and-optimize matheuristic and a full MIP model solve, including a diversification scheme.
  - A MIP-based solution method that decouples timetable development and student sectioning
  - The best solution for 29 of 30 instances during the competition and, at time of writing, for 23 instances post competition.

- A comparison of metaheuristic (simulated annealing and CP-based local search) and exact methods (MIP models and two-stage decomposition) for the IETT problem. The exact methods include the first MIP models and lower bounds in the literature and prove that 11 of 40 instances have been solved to optimality.

3.3 Practical Applications

This section discusses the practical applications. This Ph.D. study was conducted under the Industrial Ph.D. Program, making practical usability of its outcome especially relevant.

The timetabling suite of Lectio made the perturbation suggestion algorithm presented in Chapter 5 available for users in February 2020. Approximately 200 Danish high schools use the timetabling suite in practice. The algorithm has a graphical interface to help users quickly identify the consequences of each suggestion. Lectio has received positive feedback from timetablers who use it during the development of their semester timetables.

The ITC 2019 university course timetabling problem formulation comes from UniTime, which many universities use in practice. Thus, the state-of-the-art solver produced as part of this Ph.D. study has direct possible practical applications. However, the solver has not been made available to any end-users as it is still in its infancy and has no relevance to users of Lectio (high schools). Nonetheless, the solver has supported the timetabling personnel of the Technical University of Denmark to develop semester timetables during the COVID-19 pandemic. Increased student enrollments and political demand for increased distancing, resulting in reduced room capacities, made the timetabling problem especially difficult. The solver was used to produce timetables for the timetabling employees at the university.

The IETT problem represents a university examination timetabling problem found in real-world universities. Thus, although the solution methods presented in Chapter 10 have not faced practical use, they have the potential.

The perturbation suggestion algorithm and the ITC 2019 university course timetabling solver have already been used in practice to provide decision support. Although both have the potential for improvement, they have already proven helpful in practical applications.

Chapter 4

Conclusion

Educational timetabling has received much research attention throughout the years. Researchers consider the main problems of high school timetabling, university course timetabling, and university examination timetabling. Most research has focused on heuristic methodologies that are attractive in practical settings because they typically find quality solutions rapidly. However, heuristics have some disadvantages, such as being rather problem-specific and providing no bound information to evaluate solution quality. Conversely, exact methods, like MIP, can suffer from inferior performance but benefit from providing bounds. This thesis has aimed to investigate the application of MIP-based methods for practical educational timetabling problems. In practical timetabling, both speed and solution quality have great importance.

The first article submitted focused on providing dynamic timetabling support in a practical high school timetabling problem. The user selects some assignments that they would like changed, and the method finds several perturbation suggestions. The suggestions focus on improving timetable characteristics for the selected entities but consider all negative timetable consequences. The suggested method works by converting the MIP model of the problem and should be usable for other timetabling problems. We are unaware of similar approaches in the literature. We have implemented the algorithm in production code, and it is available for timetablers to develop real-world semester timetables.

A significant part of this thesis has focused on the university course timetabling posed by the ITC 2019. We detail the work on the problem in a technical report and three articles. The MIP model of the problem represents a significant contribution as it provides the first exact method for the problem. The MIP model has allowed us to produce lower bounds for all the ITC 2019 benchmark instances, and we have solved five instances to optimality. The complete solver, a parallelized matheuristic, won the competition, producing the best solution on 29 out of 30 instances. Even including the best solutions submitted after the competition, the solver has still found the best solutions on 23 instances. Thus, the parallelized matheuristic defines the state-of-the-art for the ITC 2019 problem.

The last article submitted focused on comparing exact and metaheuristic methods to the IETT, a real-world university examination timetabling problem. In the paper, we compare simulated annealing, constraint programming-based local search, directly solving two complete MIP models, and MIP-based two-stage decomposition. The simulated annealing algorithm proved to have the best average performance, closely followed by the two-stage decomposition method. In this work, we contribute the first MIP models of the given examination timetabling problem and lower bounds, proving optimality on 11 out of 40 instances.

This thesis has successfully applied MIP methods to practical educational timetabling problems covering high school timetabling, university course timetabling, and university examination timetabling. The produced work has shown that MIP-based methods have a broad application spectrum in educational timetabling and are competitive with conventional and popular heuristic techniques. Many would likely discourage using MIP-based methods in practical educational timetabling, but this project has shown that MIP indeed has merit and that MIP-based methods are worthy of study for practical applications.

4.1 Future Research

This section discusses some future research directions, focusing first on the work produced in this Ph.D. study before discussing the future of practical applications and the ITC 2019 problem.

Generally, literature on educational timetabling focuses more on heuristics solution techniques than exact methods, which is a shame since bounds are essential for evaluation solution methods. This thesis presents the first MIP models in the literature for two different educational timetabling benchmark problems. The ITC 2019 university course timetabling benchmark problem is likely to gain much interest, and therefore the MIP model of the problem will prove very useful.

We still have a large gap for many of the ITC 2019 benchmark instances. We can decrease the gap by finding better solutions or improving the bound. Undoubtedly, researchers will find improving solutions. However, future research should also investigate more advanced exact methods, potentially using our MIP model, to improve the lower bounds. We can still improve upon the graph-based MIP model by using additional graph methods on the conflict graph, but we should also investigate other methods, like various decomposition techniques. Similarly, we should investigate improvements in the MIP model of the IETT problem.

While the parallelized matheuristic solver for the ITC 2019 problem proved successful, several ways of improving it still exist. Considering the fix-and-optimize matheuristic, it could better predict a fitting neighborhood size and implement a better approach for dynamically updating it. Additionally, it could benefit from having new neighborhoods available to search.

We identify the collaboration of multiple methods as a significant strength of our ITC 2019 solver. However, in its current state, the overall algorithm control is relatively simple. In the future, we should improve on the collaboration by adding more solution strategies with different purposes and implementing more intelligent executive control. For example, we could shift the focus of the fix-and-optimize matheuristic to find many reasonable quality solutions and introduce methods like path relinking (Glover et al., 2000) and MIP polishing (Rothberg, 2007) to improve them. Such a change may yield better results more quickly, especially if we enable the algorithm to decide what methods to run dynamically. Similarly, we should investigate a more intelligent diversification scheme as the current approach is somewhat naive.

Finally, following the ITC 2019, we have investigated the resource utilization of the parallelized matheuristic and identified several possible improvements. For several of the methods, the MIP solver can use multiple CPU cores (typically four). However, in some cases, the MIP solver rarely uses more than one core since additional cores are generally only usable in the branch-and-bound phase of the search. Depending on multiple factors, the branch-and-bound phase may only be a small fraction of the total search time, resulting in overall low core utilization. Thus, future research should investigate how the method can better utilize resources, both for individual components and for the parallelized method as a whole, by dynamically controlling each parallel method. We believe there is much opportunity for massively parallelized methods where, for example, we run many single-core fix-and-optimize searches simultaneously.

4.1.1 Practical Applications

As previously discussed and commonly agreed upon, there is a gap between researchers' timetabling methods and those implemented in practical timetabling suites. This gap is a shame considering that a critical quality of OR lies in its practical applicability and potential for providing real-world benefits. It is essential that educational timetabling research finds a way out to the practitioners to help them save time and money and for researchers to get critical feedback and a better understanding of the practical problems. The research community as a whole should strive for the diffusion of optimization methods for educational timetabling. Such a task is, of course, challenging as it requires embedding the methods in complete timetabling suites with graphical user interfaces (and many more integrated systems) and expert end-users at educational timetabling institutions. Thus, the gap is most likely bridged through industrial collaboration.

The ITC2011 and ITC 2019 presents problem formats that can handle real-world high school and university course timetabling problems. Such formats are essential for bridging the aforementioned gap between academia and practical application. Although the ITC2007 examination track (track 1) and the UniTime examination benchmark problems present close to real-world problem formats for examination timetabling, it could be beneficial for the research community to develop a unified examination timetabling problem format. At least the community should make efforts to move benchmarking to more recent and relevant problems than the Carter instances from 1996.

The parallelized matheuristic that we submitted to the ITC 2019 requires much computation power and resources. Although it is becoming more common for universities to have high-performance computing centers, we should investigate how to make the solver applicable in a smaller setting. Additionally, we developed and tuned the solver to the competition setting of ten (or more) days to find the best solution. In practice, the time to produce acceptable solutions is often low (Sørensen and Stidsen, 2012). Especially since timetablers commonly interactively develop the timetable by iteratively adjusting it and re-running the optimization algorithm. Therefore, we could investigate how to adjust the parallelized matheuristic to better support dynamic timetabling.

Finally, the literature has given most research attention to the three main educational timetabling problems. However, many other types of planning and timetabling problems exist in educational institutions. Therefore, the research community and real-world educational institutions could benefit from more research on these "other" problems. Of course, these problems may suffer from having relevance only to specific institutions, but without sharing them, the community may never realize if the problems have broader appeal.

4.1.2 ITC 2019 – The Final University Course Timetabling Benchmark Problem?

The ITC 2019 problem format has proven capable of handling real-world university course timetable problems from all around the world, making the problem format very powerful and essential for the future of the research area. However, even though the competition organizers have slightly simplified it, the problem format is rather complex. Increased complexity is a consequence of representing real-world problems, but it still results in an increased entry barrier for academics wanting to tackle the problem. It took us more than a month to set up a framework for handling, parsing, and validating the ITC 2019 formats. It took equally long for Team Lectio to do the same for the ITC2011 / XHSTT format, the format for unsimplified high school timetabling problems (Sørensen, 2021).

Although the ITC 2019 presents a crucial benchmark problem that is both well documented and supported by an online validator (that also handles custom instances), one could fear that the entry barrier may discourage some academics from the problem. The university course timetabling problems of the ITC2007 (tracks 1 and 2) have received much attention, and perhaps the relatively more manageable data formats are a contributing factor. The ITC 2019 problem needs broad adoption to be valuable, and researcher hesitancy would hinder overall research progress. Thus, in the future, the research community could take steps to break down the barrier of entry, for example, by providing open-source code for handling the problem format.

One of the greatest attributes of the ITC 2019 problem is that it can encompass many different real-world university course timetabling problems. Thus, it would be regrettable to simplify the format to decrease the barrier of entry. Instead, there is potential for extending the problem formulation to include additional aspects and characteristics of educational timetabling. For example, the IETT problem considered in Chapter 10 includes the concept of composite rooms, i.e., rooms that combine into a single larger room. The ITC 2019 data format does not support such rooms. Therefore, the research community could extend the data format to model additional practical timetabling aspects and increase its real-world applicability.

Given that the ITC 2019 problem format can handle many real-world problems, the research community could consider whether it needs more university course timetabling benchmark problems in the future. Perhaps the community should instead focus on improving the ITC 2019 format and applying it to various real-world problems rather than developing new benchmark problems.

References

Barták, R., Müller, T., and Rudová, H. (2004). "A New Approach to Modeling and Solving Minimal Perturbation Problems". In: *Recent Advances in Constraints.* Ed. by K. R. Apt, F. Fages, F. Rossi, P. Szeredi, and J. Váncza. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 233–249. ISBN: 978-3-540-24662-6.

Battistutta, M., Ceschia, S., De Cesco, F., Di Gaspero, L., Schaerf, A., and Topan, E. (2020). "Local Search and Constraint Programming for a Real-World Examination Timetabling Problem". In: *17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2020).* Ed. by E. Hebrard and N. Musliu. Vol. 12296. LNCS. Springer International Publishing, pp. 69–81.

Beligiannis, G. N., Moschopoulos, C. N., Kaperonis, G. P., and Likothanassis, S. D. (2008). "Applying evolutionary computation to the school timetabling problem: The Greek case". In: *Computers & Operations Research* 35.4, pp. 1265–1280.

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., and Urli, T. (2016). "Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem". In: *Computers & Operations Research* 65, pp. 83–92. ISSN: 0305-0548. DOI: `https://doi.org/10.1016/j.cor.2015.07.002`. URL: `https://www.sciencedirect.com/science/article/pii/S0305054815001690`.

Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. (2015). "An overview of curriculum-based course timetabling". In: *Top* 23.2, pp. 313–349. DOI: `10.1007/s11750-015-0363-2`. URL: `http://dx.doi.org/10.1007/s11750-015-0366-z`.

Bixby, R. E. (2014). *Business and Mathematics: A Saga of 25 Years of Progress in Optimization.* `https://www.math.uwaterloo.ca/~hwolkowi/henry/teaching/f16/602.f16/602miscfiles/UF_Entrepreneurship_19November2014.pdf`. Accessed: 2021-07-16. Gurobi Optimization.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). "Hybrid metaheuristics in combinatorial optimization: A survey". In: *Applied soft computing* 11.6, pp. 4135–4151.

Bonutti, A., De Cesco, F., Di Gaspero, L., and Schaerf, A. (2012). "Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results". In: *Annals of Operations Research* 194.1, pp. 59–70.

Brandt, J. (1963). *Skemalægning: vejledning og orientering i skolernes årlige skemalægningsarbejde.* Gjellerup.

Burke, E. K., Newall, J. P., and Weare, R. F. (1995). "A memetic algorithm for university exam timetabling". In: *international conference on the practice and theory of automated timetabling.* Springer, pp. 241–250.

Burke, E. K., Newall, J. P., and Weare, R. F. (Mar. 1998). "Initialization Strategies and Diversity in Evolutionary Timetabling". In: *Evolutionary Computation* 6.1, pp. 81–103. ISSN: 1063-6560. DOI: `10.1162/evco.1998.6.1.81`. eprint: `https://direct.mit.edu/evco/article-pdf/6/1/81/1493044/evco.1998.6.1.81.pdf`. URL: `https://doi.org/10.1162/evco.1998.6.1.81`.

Bykov, Y. (2003). "The description of the algorithm for international timetabling competition". In: *International Timetable Competition.*

Carter, M. W., Laporte, G., and Lee, S. Y. (1996). "Examination timetabling: Algorithmic strategies and applications". In: *Journal of the operational research society* 47.3, pp. 373–383.

Carter, M. and Laporte, G. (1998). "Recent developments in practical course timetabling". eng. In: *Lecture Notes in Computer Science* 1408, pp. 3–19. ISSN: 03029743.

Cooper, T. B. and Kingston, J. H. (1996). "The complexity of timetable construction problems". In: *Practice and Theory of Automated Timetabling.* Ed. by E. Burke and P. Ross. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 281–295. ISBN: 978-3-540-70682-3.

Cordeau, J.-F., Jaumard, B., and Morales, R. (2003). "Efficient Timetabling Solution with Tabu Search Jean-François Cordeau1, Brigitte Jaumard2, 3, Rodrigo Morales2". In:

Di Gaspero, L., Mccollum, B., and Schaerf, A. (Jan. 2007). "The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3)". In:

Di Gaspero, L. and Schaerf, A. (2003). "Timetabling Competition TTComp 2002: Solver Description". In: *International Timetabling Competition.*

Dostert, M., Politz, A., and Schmitz, H. (2016). "A complexity analysis and an algorithmic approach to student sectioning in existing timetables". In: *Journal of Scheduling* 19.3, pp. 285–293.

El Sakkout, H. and Wallace, M. (2000). "Probe backtrack search for minimal perturbation in dynamic scheduling". In: *Constraints* 5.4, pp. 359–388.

Fonseca, G. H., Santos, H. G., Toffolo, T. A., Brito, S. S., and Souza, M. J. (2012). "A SA-ILS approach for the High School Timetabling Problem". In: *Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012)*, pp. 493–496.

Fonseca, G. H., Santos, H. G., and Carrano, E. G. (2016). "Late acceptance hill-climbing for high school timetabling". In: *Journal of Scheduling* 19.4, pp. 453–465.

Fukunaga, A. (2013). "An Improved Search Algorithm for Min-Perturbation". In: *Principles and Practice of Constraint Programming.* Ed. by C. Schulte. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 331–339. ISBN: 978-3-642-40627-0.

Gashi, E. and Sylejmani, K. (2019). *Simulated Annealing with Penalization for University Course Timetabling.* Available at `https://www.itc2019.org/papers/itc2019-gashi.pdf`. Accessed: 2021-07-13.

Glover, F. (1986). “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5, pp. 533–549.

Glover, F., Laguna, M., and Martí, R. (2000). “Fundamentals of scatter search and path relinking”. In: *Control and cybernetics* 29.3, pp. 653–684.

Gotlieb, C. (1962). “The construction of class-teacher timetables”. In: *Communications of the ACM.* Vol. 5. 6. ASSOC COMPUTING MACHINERY 1515 BROADWAY, NEW YORK, NY 10036, pp. 312–313.

Haan, P. de, Landman, R., Post, G., and Ruizenaar, H. (2007). “A Case Study for Timetabling in a Dutch Secondary School”. In: *Practice and Theory of Automated Timetabling VI.* Ed. by E. K. Burke and H. Rudová. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 267–279. ISBN: 978-3-540-77345-0.

Hansen, E. (1967). “Skemalægningsproblemet”. Richard Petersens Plads, Building 324, DK-2800 Kgs. Lyngby, Denmark, compute@compute.dtu.dk: Technical University of Denmark, Department of Applied Mathematics and Computer Science. URL: `http://www.compute.dtu.dk/English.aspx`.

Holm, D. S., Mikkelsen, R. Ø., Sørensen, M., and Stidsen, T. J. R. (2019). *A MIP-based approach for International Timetabling Competition 2019.* Available at `https://www.itc2019.org/papers/itc2019-holm.pdf`. Accessed: 2021-07-13.

Hoshino, R. and Fabris, I. (2020). “Optimizing Student Course Preferences in School Timetabling”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Ed. by E. Hebrard and N. Musliu. Cham: Springer International Publishing, pp. 283–299. ISBN: 978-3-030-58942-4.

INFORMS (2021). *INFORMS.* URL: `https://www.informs.org/Explore/What-is-O.R.-Analytics/What-is-O.R.` (visited on 03/18/2021).

Kingston, J. H. (2013). “Educational timetabling”. In: *Automated Scheduling and Planning.* Springer, pp. 91–108.

Kingston, J. H. (2014). “Integrated student sectioning”. In: *Practice and Theory of Automated Timetabling X, tenth international conference, PATAT 2014*, pp. 489–492.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). “Optimization by simulated annealing”. In: *science* 220.4598, pp. 671–680.

Kostuch, P. (2003). “Timetabling competition-sa-based heuristic”. In: *International Timetabling Competition.*

Kostuch, P. (2004). “The university course timetabling problem with a three-phase approach”. In: *International Conference on the Practice and Theory of Automated Timetabling.* Springer, pp. 109–125.

Kristiansen, S., Sørensen, M., and Stidsen, T. R. (2011). “Elective course planning”. In: *European Journal of Operational Research* 215.3, pp. 713–720. ISSN: 0377-2217. DOI: `https://doi.org/10.1016/j.ejor.2011.06.039`. URL: `https://www.sciencedirect.com/science/article/pii/S0377221711005686`.

Kristiansen, S. and Stidsen, T. R. (2016). "Elective course student sectioning at Danish high schools". In: *Annals of Operations Research* 239.1, pp. 99–117.

Kristiansen, S. and Stidsen, T. (2013). *A Comprehensive Study of Educational Timetabling - a Survey.* English. DTU Management Engineering Report 8.2013. DTU Management Engineering. ISBN: 978-87-93130-02-9.

Land, A. H. and Doig, A. G. (2010). "An automatic method for solving discrete programming problems". In: *50 Years of Integer Programming 1958-2008.* Springer, pp. 105–132.

Lemos, A., Monteiro, P. T., and Lynce, I. (2019). *ITC-2019: A MaxSAT approach to solve University Timetabling problems.* Available at `https://www.itc2019.org/papers/itc2019-lemos.pdf`. Accessed: 2021-07-13.

Lewis, R., Paechter, B., and Mccollum, B. (Jan. 2007). "Post Enrolment based Course Timetabling: A Description of the Problem Model used for Track Two of the Second International Timetabling Competition". In: *Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers.*

Lindahl, M., Sørensen, M., and Stidsen, T. R. (2018). "A fix-and-optimize matheuristic for university timetabling". In: *Journal of Heuristics* 24.4, pp. 645–665.

Lindahl, M., Stidsen, T., and Sørensen, M. (2019). "Quality recovering of university timetables". In: *European Journal of Operational Research* 276.2, pp. 422–435.

Lodi, A. (2013). "T rs". In: *Hybrid Metaheuristics.* Ed. by E.-G. Talbi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 273–284. ISBN: 978-3-642-30671-6. DOI: `10.1007/978-3-642-30671-6_10`. URL: `https://doi.org/10.1007/978-3-642-30671-6_10`.

McCollum, B. (Jan. 2006). "University timetabling: Bridging the gap between research and practice". In: *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling.*

McCollum, B., Mcmullan, P., Burke, E. K., Parkes, A. J., and Qu, R. (Oct. 2007). "The Second International Timetabling Competition: Examination Timetabling Track". In:

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Di Gaspero, L., Qu, R., and Burke, E. K. (2010). "Setting the research agenda in automated timetabling: The second international timetabling competition". In: *INFORMS Journal on Computing* 22.1, pp. 120–130. ISSN: 1091-9856. DOI: `10.1287/ijoc.1090.0320`.

Merlot, L. T. G., Boland, N., Hughes, B. D., and Stuckey, P. J. (2003). "A Hybrid Algorithm for the Examination Timetabling Problem". In: *Practice and Theory of Automated Timetabling IV.* Ed. by E. Burke and P. De Causmaecker. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 207–231. ISBN: 978-3-540-45157-0.

Müller, T. (Jan. 2008). "ITC2007 solver description: A hybrid approach". In: *Annals of Operations Research* 172, pp. 429–446. DOI: `10.1007/s10479-009-0644-y`.

Müller, T. (2016). "Real-life examination timetabling". In: *Journal of Scheduling* 19.3, pp. 257–270.

Müller, T. (2020). *ITC 2019: Preliminary Results Using the UniTime Solver.* Available at `https://www.unitime.org/papers/itc2019-patat2020.pdf`. Accessed: 2021-07-13.

Müller, T. and Murray, K. (2010). "Comprehensive approach to student sectioning". In: *Annals of Operations Research* 181.1, pp. 249–269.

Müller, T., Rudová, H., and Barták, R. (2005). "Minimal Perturbation Problem in Course Timetabling". In: *Practice and Theory of Automated Timetabling V.* Ed. by E. Burke and M. Trick. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 126–146. ISBN: 978-3-540-32421-8.

Müller, T., Rudová, H., and Müllerová, Z. (2018a). *University course timetabling and International Timetabling Competition 2019.* `https://www.unitime.org/present/patat18-slides.pdf`. Accessed: 2021-04-12. UniTime.

Müller, T., Rudová, H., and Müllerová, Z. (2018b). "University course timetabling and international timetabling competition 2019". In: *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria.* Ed. by E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, pp. 5–31.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). "MiniZinc: Towards a standard CP modelling language". In: *CP 2007.* Ed. by C. Bessière. Vol. 4741. LNCS. The MiniZinc toolchain is available at `https://www.minizinc.org`. Springer, pp. 529–543.

Paechter, B., Gambardella, L., and Rossi-Doria, O. (2002). *International timetabling competition 2002.* URL: `http://sferics.idsia.ch/Files/ttcomp2002/`.

Phillips, A. E., Walker, C. G., Ehrgott, M., and Ryan, D. M. (2017). "Integer programming for minimal perturbation problems in university course timetabling". In: *Annals of Operations Research* 252.2, pp. 283–304.

Pillay, N. (2010). "An overview of school timetabling research". In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pp. 321–335.

Pillay, N. (2016). *A review of hyper-heuristics for educational timetabling.* Vol. 239. 1. Springer US, pp. 3–38. DOI: `10.1007/s10479-014-1688-1`. URL: `http://dx.doi.org/10.1007/s10479-014-1688-1`.

Post, G. (2021). *Benchmarking project for (high) school timetabling.* URL: `https://www.utwente.nl/en/eemcs/dmmp/hstt/` (visited on 03/29/2021).

Post, G., Ahmadi, S., Daskalaki, S., Kingston, J. H., Kyngas, J., Nurmi, C., and Ranson, D. (2012). "An XML format for benchmarks in high school timetabling". In: *Annals of Operations Research* 194.1, pp. 385–397.

Post, G., Di Gaspero, L., Kingston, J. H., McCollum, B., and Schaerf, A. (2016). "The Third International Timetabling Competition". In: *Annals of Operations Research* 239.1, pp. 69–75. DOI: `10.1007/s10479-013-1340-5`. URL: `http://dx.doi.org/10.1007/s10479-013-1340-5`.

Post, G., Kingston, J. H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., et al. (2014). "XHSTT: an XML archive for high school timetabling problems in different countries". In: *Annals of Operations Research* 218.1, pp. 295–301.

Prida Romero, B. (1982). “Examination scheduling in a large engineering school: A computer-assisted participative procedure”. In: *Interfaces* 12.2, pp. 17–24.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., and Lee, S. Y. (2009). “A survey of search methodologies and automated system development for examination timetabling”. In: *Journal of Scheduling* 12.1, pp. 55–89. ISSN: 10946136. DOI: 10.1007/s10951-008-0077-5.

Rappos, E., Thiémard, E., Robert, S., and Hêche, J.-F. (2019). *International Timetabling Competition 2019: A Mixed Integer Programming Approach for Solving University Timetabling Problems.* Available at https://www.itc2019.org/papers/itc2019-rappos.pdf. Accessed: 2021-07-13.

Er-rhaimini, K. (2019). *Forest growth optimization for solving timetabling problems.* Available at https://www.itc2019.org/papers/itc2019-er-rhaimini.pdf. Accessed: 2021-07-13.

Rothberg, E. (2007). “An evolutionary algorithm for polishing mixed integer programming solutions”. In: *INFORMS Journal on Computing* 19.4, pp. 534–541.

Rudová, H., Müller, T., and Murray, K. (2011). “Complex university course timetabling”. In: *Journal of Scheduling* 14.2, pp. 187–207. ISSN: 10946136. DOI: 10.1007/s10951-010-0171-3. URL: https://www.researchgate.net/publication/225381506.

Sabin, G. and Winter, G. (1986). “The impact of automated timetabling on universities—A case study”. In: *Journal of the operational Research Society* 37.7, pp. 689–693.

Schaerf, A. (1999). “A survey of automated timetabling”. In: *Artificial intelligence review* 13.2, pp. 87–127.

Schindl, D. (2019). “Optimal student sectioning on mandatory courses with various sections numbers”. In: *Annals of operations research* 275.1, pp. 209–221.

Sørensen, M. (July 16, 2021). Personal communication.

Sørensen, M. and Stidsen, T. J. R. (2012). “Integer programming and adaptive large neighborhood search for real-world instances of high school timetabling”. In: *Annals of Operations Research, PATAT.*

Tan, J. S., Goh, S. L., Kendall, G., and Sabar, N. R. (2021). “A survey of the state-of-the-art of optimisation methodologies in school timetabling problems”. In: *Expert Systems with Applications* 165, p. 113943.

Terashima-Marín, H., Ross, P., and Valenzuela-Rendón, M. (1999). “Evolution of constraint satisfaction strategies in examination timetabling”. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pp. 635–642.

Vrielink, R. A. O., Schepers, D., and Jansen, E. A. (2016). “Practices in Timetabling in Higher Education Institutions”. In: *Proceedings of the 11th International Confenference on Practice and Theory of Automated Timetabling (PATAT-2016).* Udine, Italy, pp. 295–316–10. URL: https://research.utwente.nl/en/publications/practices-in-timetabling-in-higher-education-institutions.

Part II

Scientific Papers

Chapter 5

Educational Timetabling: Optimizing quality of perturbations

Rasmus Ørnstrup Mikkelsen[a,b] · Matias Sørensen[b] · Thomas Jacob Riis Stidsen[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]MaCom A/S, Vesterbrogade 48, 1., DK-1620 København V, Denmark

Status: Submitted to Annals of Operations Research

Abstract: Educational timetabling poses a classic and important problem within Operations Research. Most research focuses on the static problem of timetabling entire semesters at a time, whereas research into more dynamic methods is scarce. Consequently, many timetabling systems lack proper support for assisting practical timetablers in changing established timetables. We propose and apply an approach for supporting timetablers in making focused changes to a timetable by providing perturbation suggestions based on their input. Suggestions focus on improving the timetable for specific entities while considering negative consequences for the timetable as a whole. We apply a Mixed Integer Programming (MIP) approach to the high school timetabling problem in Denmark. We show how to adapt such a MIP model to force timetable perturbations and to prefer positive changes for user-defined timetable entities. We also introduce an approximation of the implemented model and compare the two models through a series of computational tests performed on real-world data. The tests show that the approximation provides suggestions much more quickly and of only slightly lower quality.

Keywords: Educational Timetabling · Dynamic Timetabling · Decision Support · Mixed Integer Programming

5.1 Introduction

Semester lecture timetabling is a difficult practical problem that educational institutions regularly face. Many lectures (events) must be assigned to limited resources while adhering to a number of constraints and considering timetable quality, making the timetabler's task very complicated. Not surprisingly, the application of Operations Research techniques in educational timetabling has received increasing research attention in the last few decades (Junginger, 1986; Schaerf, 1999; Lewis, 2008; Cacchiani et al., 2013; Bettinelli et al., 2015), with a number of resultant solution approaches for developing semester timetables. Nonetheless, these studies' focus has been on an academic, one-time production of a timetable (i.e., tactical timetabling) rather than supporting practical timetable development. Typically, practical educational timetablers (those developing the timetable) will use some form of software to develop a base timetable, which they then change manually—either due to modeling limitations or subjective considerations (e.g., making the timetable "better" or more "fair" for some individuals). Either way, the large number of manual changes that the timetabler must make results in very cumbersome and time-consuming work.

Compared to tactical (semester) timetabling, little research has gone into a dynamic view of educational timetabling (Lindahl et al., 2018). However, the focus of that research is primarily on the recovery problem, i.e., reaching a feasible timetable after some disruption has made an established timetable infeasible. A typical variant of the recovery problem is the minimum perturbation problem in which the objective is to regain timetable feasibility with as few lecture assignment changes as possible (El Sakkout and Wallace, 2000). Lindahl et al.'s (2019) novel approach also suggests considering the quality of the recovered timetable. Nevertheless, the goal of these approaches is to repair timetables that have become infeasible and not to support the initial timetable development.

We propose a method for supporting the timetabler to change problematic parts of a timetable, both as part of the initial development and during the semester. It is very common in practice that an educational timetable must be changed many times throughout a semester. It can be challenging to gauge the consequences of perturbing a timetable, and therefore it is demanding to maintain an educational timetable. Our method allows the timetabler to choose some lectures and provides perturbation suggestions that improve the timetable for related entities (students, teachers, and classes) while balancing global timetable deterioration.

In the method, we adjust the Mixed Integer Programming model for the timetabling problem to consider changes in quality metrics (denoted Δ-model). Input events from the timetabler define the Δ-model, such that solutions to the optimization problem correspond to perturbations in which at least one input event has changed assignment. The Δ-model objective function is defined such that good solutions to the model correspond to perturbations that balance timetable improvements for input entities and negative consequences overall. We call such a timetable perturbation a "suggestion," defined as some events that need moving to specific timeslots (and rooms).

Even if parts of a timetable could be easily improved, this method aims to support changing user-specified parts of a timetable. Therefore, our method only provides suggestions directly related to the timetablers input. The Δ-model ensures the relatedness of suggestions by constraining when to allow rewarding objective changes. The practical timetabler sees an overview of all quality metric changes for each suggestion, enabling them to more easily, and with more information, make targeted timetable changes.

Implemented in the timetabling part of Lectio (a Danish student administration system) in February 2020, the method is now accessible to approximately 200 Danish high schools and other youth education institutions. Thus, as practical timetablers are already using the method, we perform all of our computational results on real-world data. Although we apply the method to a timetabling model specific to Danish high schools, it should be applicable to many different model definitions, including other timetable and planning problems.

The remainder of this paper is organized as follows. Section 5.2 presents the timetabling problem in Danish high schools. Section 5.3 covers the modeling part of the proposed approach by defining and describing the timetabling problem model and then extending it with the Δ-model. Section 5.4 specifies the implemented algorithm, and Section 5.5 gives the results. Section 5.6 concludes.

5.2 Timetabling Problem at Danish high schools

Here we give an introduction to the general timetabling problem at Danish high schools. Each school has a number of teachers employed and some rooms available to host teaching. Students must attend classes of different subjects, and each class must have some weekly lectures. Students and teachers are preassigned to classes, and the goal is to develop a timetable such that all lectures of all classes can be held.

Each day of the week is divided into the same number of modules in which to have lectures. The combination of a day and module denotes a timeslot. A key notion of the problem is *events*, which represent lectures. Generally, each event represents a single lecture of a class, but it is possible to combine several lectures from different classes into a single event. Such a combination allows for handling cases where several classes should simultaneously use the same room, e.g., physical education and large projects. Figure 5.1 illustrates the notation.

Each event must be assigned a timeslot and room, and the timetabler can fix an event to a specific timeslot or room. A set of eligible rooms can be defined for each event and generally includes most rooms, but some classes (such as physical education and natural sciences) may require special rooms. Typically timetables are developed assuming that there is no difference between each week, e.i., classes have lectures in the same timeslots every week of the semester. It is possible to plan using two consecutive weeks, providing a larger degree of flexibility. For example, a class that needs five lectures a week could have four lectures in week one and six in the second when using a two-week planning horizon.

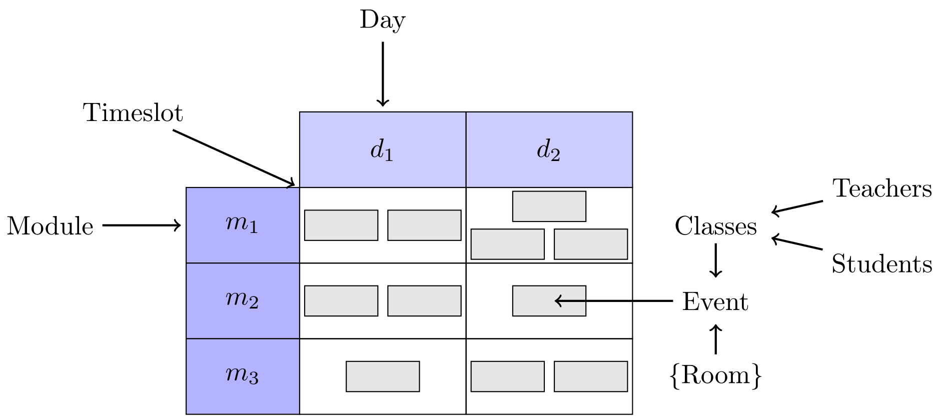


Figure 5.1: Notation used.

The timetabling problem consists of scheduling all events to a timeslot and room, such that it is possible to hold all lectures without a person (teacher/student) and room double-booking. Beyond creating a valid timetable, some quality metrics are included to measure how “good” a given timetable is. Following is an overview of the quality metrics included in the problem.

Event assignment The event assignment penalty is separated to consider timeslot and room assignment individually. Earlier modules in the day are preferred (penalized less). However, teachers can designate personal “undesired” timeslots, which are considered in the final timeslot penalty for a specific event and timeslot pair. Each event has room priorities defined in three tiers, such that assignments to lower priority rooms give a larger penalty than assignments to higher priority rooms. For events locked to a timeslot or room, the respective assignment penalty is 0.

Idle timeslots It is desirable to avoid idle timeslots for students and teachers. Idle timeslots are timeslots where a person has no events scheduled but has events earlier and later timeslots in that day. Idle timeslots must be kept minimal for students and less so for teachers, as reflected in a higher penalty for each idle timeslot for students.

Class room stability It is preferred that a class has all of its lectures in the same room. Therefore we want to minimize the number of “excess” rooms used by events of the same class. Therefore a penalty is applied for each room a class is assigned except the first one.

Class neighbor days To allow for more time for preparation and homework, it is best to spread out lectures of the same class throughout the week. Therefore we would like to avoid neighbor day clashes, where lectures of the same class are held on contiguous days. We apply a penalty for each neighbor day pair (Monday-Tuesday, Tuesday-Wednesday, etc.) where a class has lectures on both days, unless the class is locked into at least one event on the two days.

Teacher workload Teachers prefer a balanced workload on the days of the week. While there is a limit on the maximum number of events for a teacher on a single day, teachers would also like to avoid days with too few events. Thus on days where the teacher has events scheduled, we prefer that the teacher has at least two. Therefore a penalty is incurred for any day where a teacher only has one event scheduled.

Teacher days off Not only is it preferable for teachers to avoid days with only one event, but they would also like to have as many days off as possible. We impose a teacher dependent hard lower limit on the number of days off. However, days off is also included as a quality metric to incentivize as many days off as possible. To reward teacher days off, we include a penalty for each day that a teacher has events scheduled.

Student days off As opposed to teachers, it is desirable that students do not have days off. A penalty is included for each day that a student does not have any scheduled events.

Class and teacher week lecture stability When producing a timetable using a two-week planning horizon, we prefer a balanced distribution of events for each class and teacher. We apply a penalty for the absolute difference in the number of events scheduled for a class or teacher each week. A difference of 1 incurs no penalty as it is unavoidable when the class or teacher has an odd number of events.

Deviation from the previous solution It is common for timetablers to tweak timetables during the initial timetable development and throughout the school year. When given a previous solution, we should not make drastic changes to it. So when an already establish timetable is used as a starting point, we include a small penalty for each event assignment that deviates in timeslot or room compared to the previous solution.

5.3 Solution approach

This method aims to provide perturbation suggestions focused on improving the timetable for entities associated with a given number of input events. First, we consider an example to showcase the definition of a suggestion and what types of suggestions to avoid. A timetable is shown in Figure 5.2 where each box denotes an event. We focus on two students, where the first take part in the green and blue events, and the second take part in the red events. The grey boxes are other events, which neither of the two students attends. In this example, at most one event can be scheduled in a timeslot. On Monday and Wednesday, the first student has 1 and 2 idle timeslots, respectively, and the second student has 2 idle timeslots on Thursday.

The user gives e_2 as input (shown in blue), and a valid suggestion would be to move it to module 2 and "push" the other event away, decreasing the number of idle timeslots for the first student. We would like to avoid suggesting moves such as the one shown for the second student. While this suggestion improves the overall solution (eliminates two idle timeslots), it cannot be deemed a consequence of the given input and should not be suggested. Likewise, we should not suggest the move shown for event e_4 as this is also not a consequence of the input, even though the events share an input student. In this case, the best suggestion would be to move both events e_1 and e_2 to the second and third modules on Wednesday, eliminating a total of three idle timeslots for the first student. Although e_1 is not an input event, moving this event in combination with e_2 can undoubtedly be deemed as a consequence of the given input.

In this work, we say a move is **related** when it can be considered a rational consequence of the input events and thus is allowed as a suggestion.

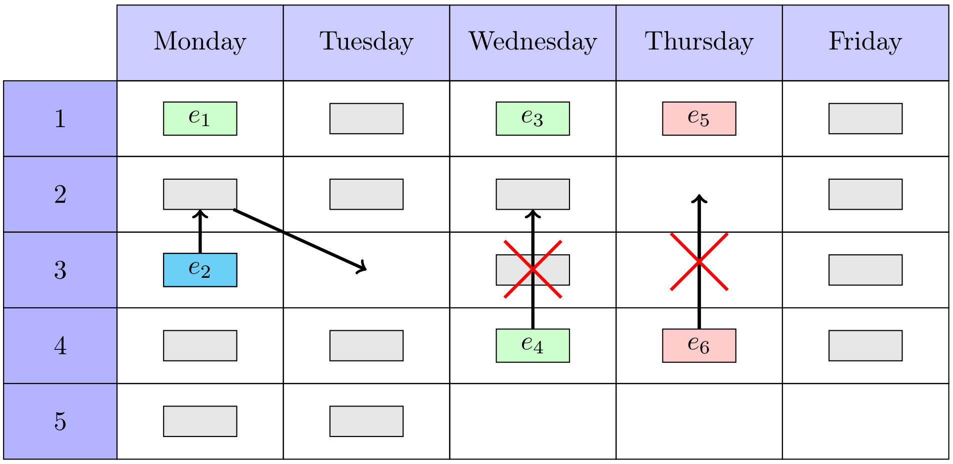


Figure 5.2: A small example showing valid and non-valid suggestions.

Our method is applied to the timetabling problem as defined in Lectio. Section 5.3.1 gives a brief introduction of the MIP formulation of this problem. A more detailed description is available in Sørensen and Stidsen, 2013. In Section 5.3.2 this MIP model is extended to focus on quality metric changes, resulting in the Δ-model, which allows us to find perturbation suggestions.

5.3.1 Basic model

Here we give a brief introduction to the MIP used in Lectio, defining the most important sets, parameters, and variables. The model is defined to allow for *not* assigning an event to a timeslot and/or room, ensuring there always exists a feasible solution. Having unassigned events is associated with a big cost and is in practice extremely unlikely in an optimal (or reasonably good) solution. In addition, it is possible to define the problem with and without rooms. In general, room availability is not a problem for Danish high schools, and timetablers sometimes prefer to assign each event to timeslots first and, subsequently, assign rooms. We denote the dummy timeslot and room t_D and r_D, respectively.

Sets	**Description**
$\mathcal{D}$	The set of days
$\mathcal{M}$	The set of modules
$\mathcal{T}$	The set of timeslots (combination of days and modules)
$\mathcal{E}$	The set of events
$\mathcal{R}$	The set of rooms
$\mathcal{C}$	The set of classes
$\mathcal{A}$	The set of persons (students and teachers)
$\mathcal{A}_t$	The set of teachers ($\mathcal{A}_t \subset \mathcal{A}$)
$\mathcal{A}_s$	The set of students ($\mathcal{A}_s \subset \mathcal{A}$)
Parameters	
M_a	The number of persons represented by person a
B_{ea}	1 if person a is part of event e
J_{ec}	1 if class c is part of event e
G_{rt}	1 if room r is available in timeslot t
LT_{et}	1 if event e is locked into timeslot t
LR_{er}	1 if event e is locked into room r
K_{er}	1 of event e can use room r
Helper	
$\mathcal{T}_d$	The set of timeslots belonging to day d
d_t	The day containing timeslot t
$t_P(e)$	Previous timeslot assignment of event e (given prev. solution)
$r_P(e)$	Previous room assignment of event e (given prev. solution)

Table 5.1: Some of the sets, parameters and helper functions used in the MIP model.

The sets and parameters used in the MIP formulation are shown in Table 5.1. We reduce the problem size by grouping students attending the exact same events into a super-entity. This introduces parameter $M_a \in \mathbb{N}$ to denote the number of 'real' persons which person a represents. Person grouping is not done for teachers, so $M_a = 1 \; \forall a \in \mathcal{A}_t$.

The main decision variable is $x_{ert} \in \{0, 1\}$, which takes a value of 1 when event e is assigned to room r in timeslot t. An auxiliary variable $y_{et} \in \{0, 1\}$ is defined to take a value of 1 when event e is assigned timeslot t. Including this variable greatly simplifies the description of certain constraints and reduces the number of non-zeros of the model. Additionally, this variable functions as the main decision variable when defining the model without rooms. Each event is assigned a room and time using constraint (5.1) and y_{et} is set using constraint (5.2). Remember that it is possible to assign events to a dummy timeslot and dummy room.

$$\begin{aligned} \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} x_{ert} &= 1 \quad \forall e \in \mathcal{E} && (5.1) \\ \sum_{r \in \mathcal{R}} x_{ert} &= y_{et} \quad \forall e \in \mathcal{E}, t \in \mathcal{T} && (5.2) \end{aligned}$$

When rooms are not included, the left hand side of constraint (5.1) is changed to $\sum_{t \in \mathcal{T}} y_{et}$ and constraint (5.2) is dropped.

It is not allowed to double-book either persons or rooms. Thereby each person can only attend one event in any timeslot (except the dummy timeslot). Likewise, each room (except the dummy room) can only be assigned once to any timeslot (except the dummy timeslot). Additionally, some rooms may be unavailable in specific timeslots. Avoiding person double-booking is handled by constraint (5.3) and room double-booking and (un)availability is observed by constraint (5.4).

$$\begin{aligned} \sum_{e \in \mathcal{E}} B_{ea} y_{et} &\le 1 \quad \forall a \in \mathcal{A}, t \in \mathcal{T} \setminus \{t_D\} && (5.3) \\ \sum_{e \in \mathcal{E}} x_{ert} &\le G_{rt} \quad \forall r \in \mathcal{R} \setminus \{r_D\}, t \in \mathcal{T} \setminus \{t_D\} && (5.4) \end{aligned}$$

It if possible for the user to lock events to specific rooms and/or timeslots, which is observed by imposing constraints (5.5) and (5.6) respectively.

$$\begin{aligned} y_{et} &= 1 \quad \forall e \in \mathcal{E}, t \in \mathcal{T} : LT_{et} = 1 && (5.5) \\ \sum_{t \in \mathcal{T}} x_{ert} &= 1 \quad \forall e \in \mathcal{E}, r \in \mathcal{R}, t \in \mathcal{T} : LR_{er} = 1 && (5.6) \end{aligned}$$

Events may also require special rooms, e.g., laboratories or sports halls. Observing special room requirements is ensured by constraint (5.7).

$$\sum_{t \in \mathcal{T}} x_{ert} \le K_{er} \quad \forall e \in \mathcal{E}, r \in \mathcal{R} \qquad (5.7)$$

Finally, it is not allowed to assign an event to a room unless it is also assigned a timeslot. Conversely, it is acceptable to assign an event to a timeslot without a room. Constraint (5.8) imposes that events locked to a room are allowed to be assigned to the dummy timeslot. However, when not locked to a room, it can only be assigned a non-dummy room if it is simultaneously assigned a non-dummy timeslot.

$$\sum_{r \in \mathcal{R} \setminus \{r_D\}} x_{e,r,t_D} - \sum_{r \in \mathcal{R}} LR_{er} \le 0 \quad \forall e \in \mathcal{E} \qquad (5.8)$$

The described constraints are a brief introduction to the MIP model used in Lectio. The actual model is more complex to allow for more advanced features of the Lectio interface, e.g., grouping events together in “EventChains” and allowing specific room conflicts. EventChains are used to define that a set of events need to either take place in the same or contiguous timeslots or in the same room simultaneously.

The objective function of the basic MIP is shown in (5.9) and the variables and cost coefficients used for each quality metric is shown in Table 5.2. Here we omit the constraints for setting the quality metric related variables. The combined introduction of the basic feasible model and quality metrics provides a sufficient understanding for this paper. Once more, Sørensen and Stidsen, 2013 gives a full description of the model.

Variables	**Description**
$x_{ert} \in \{0,1\}$	1 if event e takes place in room r in timeslot t
$y_{et} \in \{0,1\}$	1 if event e takes place in timeslot t
$h_{ad} \in \mathbb{N}_0$	The number of idle slots for person a on day d
$s_c \in \mathbb{N}_0$	The number of excess rooms assigned to class c
$n_{cd} \in \{0,1\}$	1 if class c has a neighbor-day-conflict on day d
$o_{ad} \in \{0,1\}$	1 if teacher a has only one lecture on day d
$f_{ad} \in \{0,1\}$	1 if person a has day d off
$w_c \in \mathbb{N}_0$	The number of events out of week-balance for class c
$l_a \in \mathbb{N}_0$	The number of days off out of week-balance for teacher a
$u_e \in \{0,1\}$	1 if event e is not assigned to its previous timeslot
$p_e \in \{0,1\}$	1 if event e is not assigned to its previous room
Costs	
α_{ert}	Cost of assigning event e to room r in timeslot t (very high for t_D and r_D, 0 when locked)
β_a	Cost of each idle timeslot for person a
ϵ	The cost of each excess room assigned to a class
ζ	The cost of each neighbor-day-conflict
η	The cost of each day a teacher has only one lecture
γ	The cost of a teacher **not** having a day off
δ	The cost of a student having a day off
ι	The cost of each day out of week-balance for a class
τ	The cost of each day out of week-balance for a teacher
θ	The cost of each event that is not assigned to its previous timeslot or room

Table 5.2: The variables and cost coefficients used for model quality metrics.

$$
\begin{aligned}
\min & \overbrace{\sum_{e\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{t\in\mathcal{T}}\alpha_{ert}x_{ert}}^{\text{Assignment}} + \overbrace{\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} M_a\beta_a h_{ad}}^{\text{idle timeslots}} + \overbrace{\epsilon\sum_{c\in\mathcal{C}} s_c}^{\text{room stability}} + \overbrace{\zeta\sum_{c\in\mathcal{C}}\sum_{d\in\mathcal{D}} n_{cd}}^{\text{neighbor days}} \\
& + \overbrace{\eta\sum_{a\in\mathcal{A}_t}\sum_{d\in\mathcal{D}} M_a o_{ad}}^{\text{teacher workload}} + \overbrace{\gamma\sum_{a\in\mathcal{A}_t}\left(|\mathcal{D}| - \sum_{d\in\mathcal{D}} f_{ad}\right)}^{\text{teacher days off}} + \overbrace{\delta\sum_{a\in\mathcal{A}_s}\sum_{d\in\mathcal{D}} M_a f_{ad}}^{\text{student days off}} \\
& + \overbrace{\iota\sum_{c\in\mathcal{C}} w_c}^{\text{class lecture stability}} + \overbrace{\tau\sum_{a\in\mathcal{A}_t} l_a}^{\text{teacher lecture stability}} + \overbrace{\theta\sum_{e\in\mathcal{E}}(u_e + p_e)}^{\text{deviation from previous solution}}
\end{aligned}
\tag{5.9}
$$

5.3.2 Δ-model

In this section, we define the Δ-model, which considers changes in the original objectives. We are given a solution to the original model and, thus, solution values for all variables (denoted with an asterisk).

The main idea behind the Δ-model is to adjust the objective to consider absolute quality metric changes and imposing rules for when changes have weight in the objective function. These rules are defined such that "good" changes in objective variables (decrease as this is a minimization problem) only have weight in the objective function when the change can be deemed a consequence of the input events. We achieve this by introducing auxiliary variables bounded by the absolute change in variable values and imposing non-negativity constraints for specific conditions.

First, we define some general constraints for the Δ-model before extending it with quality metric specific constraints.

We are given a set of user selected events $\mathcal{E}_m \subseteq \mathcal{E}$ to consider for generating perturbation suggestions. Thus the sets of related persons $\mathcal{A}_m$ and related classes $\mathcal{C}_m$ are defined as $\mathcal{A}_m = \{\, a \in \mathcal{A} \mid (\exists e \in \mathcal{E}_m)\,[B_{ea} = 1] \,\}$ and $\mathcal{C}_m = \{\, c \in \mathcal{C} \mid (\exists e \in \mathcal{E}_m)\,[J_{ec} = 1] \,\}$ respectively. Additionally, let $\mathcal{E}_m^a$ be the set of events in $\mathcal{E}_m$ that person a is attending (i.e., $\mathcal{E}_m^a = \{\, e \in \mathcal{E}_m \mid B_{ea} = 1 \,\}$) and let $\mathcal{E}_m^c$ be the set of events in $\mathcal{E}_m$ that class c is attending (i.e., $\mathcal{E}_m^c = \{\, e \in \mathcal{E}_m \mid J_{ec} = 1 \,\}$).

We introduce the common variables and constraints of the Δ-model by considering the example of idle timeslots. We capture the total number of idle timeslots for person a on day d using the non-negative integer variable h_{ad}. The absolute change in idle timeslots for a person-day pair compared to the previous solution is then

$$\Delta_{h_{ad}} = h_{ad} - h_{ad}^*$$

Thus if $\Delta_{h_{ad}}$ is positive, then the number of idle timeslots went up, and conversely, if the value is negative, it went down. In the Δ-model we need to consider all idle timeslot changes, but we wish to define specific rules for when to allow rewarding a decrease in idle timeslots. Negative $\Delta_{h_{ad}}$ should only be allowed when the change can be considered a consequence of the user input. To allow for this, we introduce an auxiliary integer variable $\Delta'_{h_{ad}}$, which is set using the following constraints.

$$\begin{aligned} \Delta'_{h_{ad}} &\geq \Delta_{h_{ad}} && \forall a \in \mathcal{A}, d \in \mathcal{D} \\ \Delta'_{h_{ad}} &\geq 0 && \forall a \in \mathcal{A} \setminus \mathcal{A}_m, d \in \mathcal{D} \end{aligned}$$

And we add the following term to the objective function

$$\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} M_a \beta_a \Delta'_{h_{ad}}$$

As this is a minimization problem, the first constraint ensures that $\Delta'_{h_{ad}}$ is bounded by the absolute change in idle timeslots. However, for people that are guaranteed not to be related ($a \notin \mathcal{A}_m$), $\Delta'_{h_{ad}}$ is additionally bounded by 0 by the second constraint. So for all guaranteed not related moves, the associated $\Delta'_{h_{ad}}$ may only take non-negative values, and these moves can therefore **never** be rewarding (regarding idle timeslots) while they still may provide a penalty. Thereby, even if a non-related person has a decrease in idle timeslots, this improvement is not reflected in the objective function.

We do this general procedure for all quality metrics of the model: define a Δ' variable which is at least the actual change and force non-negative for non-related (not included in the input) entities. Person days off is split into teachers and students using $\Delta'^{t}_{f_{ad}}$ and $\Delta'^{s}_{f_{ad}}$, respectively.

Notice that we do not enforce constraints such that events cannot move but simply define when moves may provide a reward. Therefore, it is acceptable that events are moved "out of the way" to avoid causing new penalties.

These generic rules are not sufficient to ensure that we only reward moves that are a consequence of the given input. Consider the example shown in Figure 5.3 where there are four events which include the same student, and event e_2 is the input. Then a valid and rewarding suggestion, given the described rules, would be to move e_2 to (d_1, m_2) and event e_4 to (d_2, m_2). However, moving event e_4 in parallel cannot be considered a direct consequence of moving e_2 individually and should not be suggested. To avoid these types of suggestions the move should not be rewarding, and therefore additional constraints are needed for this quality metric (as well as most others) to ensure desired behavior. These are defined in Section 5.3.2.2

As mentioned in Section 5.3.1, it is possible to define the problem without including rooms. In the implemented algorithm, we exclude all rooms as this saves defining all x_{ert} variables, and rooms are only used for part of the assignment and the class excess room quality metrics. Additionally, it is more valuable to provide timetablers with suggestions that move events between timeslots as the problem's temporal aspect is usually more complicated. Also, many essentially equal solutions exist by simply changing rooms. Therefore, the focus is on providing suggestions regarding changing timeslots, as it is more important and avoids much complexity.

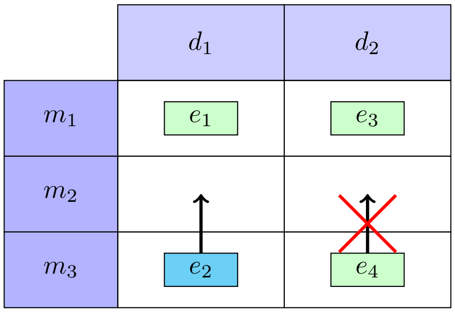


Figure 5.3: Example showing that the generic Δ' rules are not sufficient.

5.3.2.1 Δ-model constraints

Beyond the quality metric related constraints, some constraints are introduced to the model to generate valuable suggestions. Firstly, to ensure that the timetable changes, we add a "force-move" constraint. This constraint (5.10) enforces that at least one of the input events are moved from their current timeslot. Recall that $t_P(e)$ is the previous timeslot assignment of event e.

$$\sum_{e \in \mathcal{E}_m} y_{e t_P(e)} \leq |\mathcal{E}_m| - 1 \tag{5.10}$$

From a model perspective, each event is a unique entity, but in practice, there are many equivalent events, e.g., the lectures of a class where the exact same teachers and students attend. Consider the example illustrated in Figure 5.4, where each box represents a lecture of the same class. The two shown suggestions are in practice equivalent, as they both result in a lecture in (d_1, m_1) and (d_3, m_2), even though, from a modeling perspective, they are different. To avoid making such suggestions, we add constraint (5.11), which enforces that all events cannot move to a timeslot, in which an equivalent event is assigned. The set $Eq(e)$ is the set of events which are equivalent to e (excluding e). We define that two events are equivalent if they contain precisely the same teachers and students.

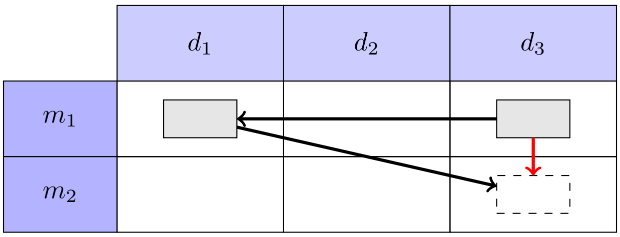


Figure 5.4: The moves shown in red and black result in the same timetable.

$$\sum_{e' \in Eq(e)} y_{et_P(e')} \leq 0 \qquad \forall e \in \mathcal{E} \tag{5.11}$$

The algorithm builds the MIP ignoring rooms, but rooms may already have been assigned to events in the given timetable. Therefore an additional constraint needs to be added to ensure that events assigned to the same room cannot be scheduled simultaneously (unless that is already the case through EventChains). Let $\mathcal{E}_r$ be the set of events assigned to room r and $\mathcal{E}_{rt}$ be the set of events assigned to room r and timeslot t, i.e., $\mathcal{E}_r = \{\, e \in \mathcal{E} \mid r_P(e) = r \,\}$ and $\mathcal{E}_{rt} = \{\, e \in \mathcal{E} \mid r_P(e) = r \wedge t_P(e) = t \,\}$. Then constraint (5.12) enforces that events using the same room can only be scheduled in the same timeslot if that is already the case in the given timetable.

$$\sum_{e \in \mathcal{E}_r} y_{et} \leq \max\{1, |\mathcal{E}_{rt}|\} \qquad \forall r \in \mathcal{R}, t \in \mathcal{T} \tag{5.12}$$

Lastly, since it is very costly to assign events to the dummy timeslot, there is incentive to move unassigned events into the timetable. To ensure that such a move is a direct consequence of the input, this should only be allowed unassigned events that are part of the user input. We enforce this restriction by fixing all unassigned non-input events to the dummy timeslot, as shown in (5.13).

$$y_{et} = 1 \qquad \forall e \in \mathcal{E} \setminus \mathcal{E}_m : t_P(e) = t_D \tag{5.13}$$

5.3.2.2 Quality metric constraints

This section specifies quality metric related constraints included in the model to cover when the generic constraints are not sufficient. These are very problem specific and may require subjective decisions on what should be allowed and rewarded. For completeness, we go through each quality metric and show all constraints.

Some constraints require information about whether a person $a \in \mathcal{A}_m$ has input events already scheduled on or moved to specific days. For this, we define parameter T_{ad} to be 1 for all days $d \in \mathcal{D}$ where person a has events in $\mathcal{E}_m^a$ assigned in the given solution. Additionally, we introduce binary variable g_{ad} to take a value of 1 if person a is assigned to day d with an event $e \in \mathcal{E}_m^a$. Variable g_{ad} is bounded using constraint (5.14). The addition of T_{ad} and g_{ad} allows us to include the movement of events $e \in \mathcal{E}_m^a$ in the model.

$$g_{ad} \leq \sum_{e \in \mathcal{E}_m^a} \sum_{t \in \mathcal{T}_d} y_{et} \qquad \forall a \in \mathcal{A}_m, d \in \mathcal{D} \tag{5.14}$$

Assignment penalty Events not chosen by the user ($\{\mathcal{E} \setminus \mathcal{E}_m\}$) should only be moved if it is beneficial to do so due to other quality metrics than assignment penalty. Therefore negative $\Delta'_{y_{et}}$ values should only be allowed for events in $\mathcal{E}_m$. Otherwise, it could be rewarding to move unrelated events regardless of the input. Thereby no additional constraints are needed for this quality metric and constraints (5.15) and (5.16) are sufficient to set $\Delta'_{y_{et}}$.

$$\begin{aligned} \Delta'_{y_{et}} &\geq \Delta_{y_{et}} && \forall e \in \mathcal{E}, t \in \mathcal{T} && (5.15)\\ \Delta'_{y_{et}} &\geq 0 && \forall e \in \mathcal{E} \setminus \mathcal{E}_m, t \in \mathcal{T} && (5.16) \end{aligned}$$

Idle timeslots An input event $e \in \mathcal{E}_m$ may be the cause of idle timeslots on the day it is scheduled and may help reduce idle timeslot on the day it is moved to. Therefore, for persons associated with events to move, negative $\Delta'_{h_{ad}}$ values should only be allowed on the days their events to move ($\mathcal{E}^a_m$) are already assigned or moved to. Thus $\Delta'_{h_{ad}}$ should be allowed to have a negative value on days where $T_{ad} + g_{ad} \geq 1$. The maximum possible number of idle timeslot on any day is determined by the number of modules: $|\mathcal{M}| - 2$. In the extremely unlikely case that the timetable has two or fewer modules per day, it is impossible to have idle timeslots, and we exclude this quality metric. Constraints (5.17) - (5.19) are thereby sufficient to set $\Delta'_{h_{ad}}$ correctly.

$$\begin{aligned} \Delta'_{h_{ad}} &\geq \Delta_{h_{ad}} && \forall a \in \mathcal{A}, d \in \mathcal{D} && (5.17)\\ \Delta'_{h_{ad}} &\geq 0 && \forall a \in \mathcal{A} \setminus \mathcal{A}_m, d \in \mathcal{D} && (5.18)\\ \Delta'_{h_{ad}} &\geq -\left(|\mathcal{M}| - 2\right)\left(T_{ad} + g_{ad}\right) && \forall a \in \mathcal{A}_m, d \in \mathcal{D} && (5.19) \end{aligned}$$

Returning to the example in Figure 5.3, the shown suggestions has $T_{ad_1} = g_{ad_1} = 1$ and $T_{ad_2} = g_{ad_2} = 0$ and constraint (5.19) thus makes the shown perturbation of event e_4 non-rewarding. Since the deviation from previous solution quality metric is included, such a move incurs a cost and is not suggested.

Neighbor days When moving an event, it is only possible to improve the neighbor day conflict count if the moved event causes such a conflict. Thus negative values for $\Delta'_{n_{cd}}$ should only be allowed for classes in $\mathcal{C}_m$ on days where their events in $\mathcal{E}^c_m$ were previously assigned **and** caused neighbor day conflicts. We denote this set of days $\mathcal{D}^c_m$. For a given class in $\mathcal{C}_m$ and a given day in $\mathcal{D}^c_m$, the neighbor day conflict count can at most decrease by 1, and on all other days, no decrease is allowed. Thereby constraints (5.20) - (5.22) correctly set the value of $\Delta'_{n_{cd}}$.

$$\begin{aligned} \Delta'_{n_{cd}} &\geq \Delta_{n_{cd}} && \forall c \in \mathcal{C}, d \in \mathcal{D} && (5.20)\\ \Delta'_{n_{cd}} &\geq 0 && \forall c \in \mathcal{C} \setminus \mathcal{C}_m, d \in \mathcal{D} && (5.21)\\ \Delta'_{n_{cd}} &\geq 0 && \forall c \in \mathcal{C}_m, d \in \mathcal{D} \setminus \mathcal{D}^c_m && (5.22) \end{aligned}$$

Teacher less than 2 lectures/day When moving an event, it is possible for the associated teacher(s) to remove or gain a day with less than two lectures on the day the event is moved from and the day it is moved to. Thus negative $\Delta'_{o_{ad}}$ should be allowed these two days. Constraints (5.23) - (5.25) are used to set $\Delta'_{o_{ad}}$.

$$\Delta'_{o_{ad}} \geq \Delta_{o_{ad}} \qquad \forall a \in \mathcal{A}_t, d \in \mathcal{D} \tag{5.23}$$
$$\Delta'_{o_{ad}} \geq 0 \qquad \forall a \in \mathcal{A}_t \setminus \mathcal{A}^t_m, d \in \mathcal{D} \tag{5.24}$$
$$\Delta'_{o_{ad}} \geq -T_{ad} - g_{ad} \qquad \forall a \in \mathcal{A}^t_m, d \in \mathcal{D} \tag{5.25}$$

Teacher days off It is desirable to have as many days off as possible for teachers. When moving an event, it allows for the associated teacher to potentially having that day off. Therefore $\Delta'^t_{f_{ad}}$ should be allowed to be negative on days where $T_{ad} = 1$. To allow for this, we impose a non-negative bound on all days where $T_{ad} = 0$. Constraints (5.26) - (5.28) are used to set $\Delta'^t_{f_{ad}}$. Notice that the right hand side of (5.26) is $f^*_{ad} - f_{ad}$ in order to set $\Delta'^t_{f_{ad}}$ correctly. For example for days where the teacher goes from **not** having the day off ($f^*_{ad} = 0$) to having the day off ($f_{ad} = 1$), then $\Delta'^t_{f_{ad}} = -1$ as $\Delta'^t_{f_{ad}}$ is added directly to the minimization objective.

$$\Delta'^t_{f_{ad}} \geq -\Delta^t_{f_{ad}} \qquad \forall a \in \mathcal{A}_t, d \in \mathcal{D} \tag{5.26}$$
$$\Delta'^t_{f_{ad}} \geq 0 \qquad \forall a \in \mathcal{A}_t \setminus \mathcal{A}^t_m, d \in \mathcal{D} \tag{5.27}$$
$$\Delta'^t_{f_{ad}} \geq 0 \qquad \forall a \in \mathcal{A}^t_m, d \in \mathcal{D} : T_{ad} = 0 \tag{5.28}$$

Student days off As opposed to teachers, it is preferred that students have no days off. It should only be possible to decrease the number of days off for a student by moving events in $\mathcal{E}_m$ to days where the student does not have any events assigned. Therefore $\Delta'^s_{f_{ad}}$ should be able to be negative on days where $g_{ad} = 1$ and $T_{ad} = 0$. Constraints (5.29) - (5.31) are used to set $\Delta'^s_{f_{ad}}$.

$$\Delta'^s_{f_{ad}} \geq \Delta^s_{f_{ad}} \qquad \forall a \in \mathcal{A}_s, d \in \mathcal{D} \tag{5.29}$$
$$\Delta'^s_{f_{ad}} \geq 0 \qquad \forall a \in \mathcal{A}_s \setminus \mathcal{A}^s_m, d \in \mathcal{D} \tag{5.30}$$
$$\Delta'^s_{f_{ad}} \geq -g_{ad} \qquad \forall a \in \mathcal{A}^s_m, d \in \mathcal{D} : T_{ad} = 0 \tag{5.31}$$

Class week lecture stability For any class $c \in \mathcal{C}_m$, suggestions can only improve their week lecture stability count if the class has input events in the week with surplus lectures. Assuming a large enough imbalance, moving a single event from the week with many events to the other will decrease the instability by two. Moving two events will cause a decrease of four, three a decrease of six, and so on. Let $\mathcal{E}_m^{c+}$ be the set of events in $\mathcal{E}_m^c$ which are in the week with more events. Let $\mathcal{T}^{c-}$ be the timeslots of the week with fewer lectures for class c. Then $\sum_{e\in\mathcal{E}_m^{c+}}\sum_{t\in\mathcal{T}^{c-}} y_{et}$ expresses the number of events in $\mathcal{E}_m^{c+}$ which are to the other week. Thus constraints (5.32) - (5.34) are used to set Δ'_{w_c}.

$$\Delta'_{w_c} \geq \Delta_{w_c} \qquad \forall c \in \mathcal{C} \tag{5.32}$$
$$\Delta'_{w_c} \geq 0 \qquad \forall c \in \mathcal{C} \setminus \mathcal{C}_m \tag{5.33}$$
$$\Delta'_{w_c} \geq -2 \sum_{e\in\mathcal{E}_m^{c+}} \sum_{t\in\mathcal{T}^{c-}} y_{et} \qquad \forall c \in \mathcal{C}_m \tag{5.34}$$

Teacher week lecture stability Teacher week lecture stability follows the same observations as for class week lecture stability; the stability count can only improve if the teacher has input events in the week with surplus lectures. Let $\mathcal{A}_m^{a+}$ be the set of events in $\mathcal{E}_m^a$ which are in the week with more events and let $\mathcal{T}^{a-}$ be the timeslots of the week with fewer lectures for teacher a. Then constraints (5.35) - (5.37) are used to set the value of Δ'_{l_a}.

$$\Delta'_{l_a} \geq \Delta_{l_a} \qquad \forall a \in \mathcal{A}_t \tag{5.35}$$
$$\Delta'_{l_a} \geq 0 \qquad \forall a \in \mathcal{A}_t \setminus \mathcal{A}_m^t \tag{5.36}$$
$$\Delta'_{l_a} \geq -2 \sum_{a\in\mathcal{A}_m^{a+}} \sum_{t\in\mathcal{T}^{a-}} y_{et} \qquad \forall a \in \mathcal{A}_m^t \tag{5.37}$$

5.3.2.3 Objectives

For the Δ-model to work, we need to set the values of the associated variables in the original model, e.g., to set $\Delta'_{h_{ad}}$ the value of h_{ad} must also be set. Therefore all the original objectives remain, and we add the Δ objectives. To ensure that the solution to the model reflects the Δ objectives, the objectives are defined in four different tiers of distinct weight levels such that each subsequent level outweighs the previous. All original objectives have their coefficients changed to 1 to ensure that the required variables have their values set correctly while maintaining their influence small. Next comes the deviation from the previously assigned timeslot, such that changes are costly enough that they are not done to improve the original objectives, but only the Δ objectives. The third tier consists of the Δ objectives, and the fourth tier with the highest weight is given to the assignment of events to the dummy timeslot. The last weight tier ensures that it is never more profitable to unassign events. Figure 5.5 illustrates the four objective weight tiers.

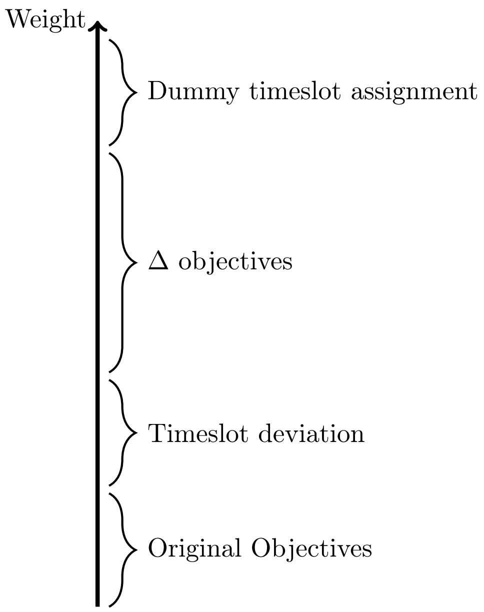


Figure 5.5: Illustrating the objective weight tiers included in the Δ-model.

The Δ objectives added are shown in (5.38) where ω_Δ is the objective coefficient multiplier for the Δ objectives.

$$\begin{aligned}\min \quad \omega_\Delta \Bigg(& \overbrace{\sum_{e\in\mathcal{E}}\sum_{t\in\mathcal{T}} \alpha_{et}\Delta'_{y_{et}}}^{\text{Assignment}} + \overbrace{\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} M_a\beta_a\Delta'_{h_{ad}}}^{\text{idle timeslots}} + \overbrace{\zeta\sum_{c\in\mathcal{C}}\sum_{d\in\mathcal{D}}\Delta'_{n_{cd}}}^{\text{neighbor days}} \\ & \overbrace{+\eta\sum_{a\in\mathcal{A}_t}\sum_{d\in\mathcal{D}} M_a\Delta'_{o_{ad}}}^{\text{teacher workload}} + \overbrace{\gamma\sum_{a\in\mathcal{A}_t}\sum_{d\in\mathcal{D}}\Delta'^{t}_{f_{ad}}}^{\text{teacher days off}} + \overbrace{\delta\sum_{a\in\mathcal{A}_s}\sum_{d\in\mathcal{D}} M_a\Delta'^{s}_{f_{ad}}}^{\text{student days off}} \\ & + \overbrace{\iota\sum_{c\in\mathcal{C}}\Delta'_{w_c}}^{\text{class lecture stability}} + \overbrace{\tau\sum_{a\in\mathcal{A}_t}\Delta'_{l_a}}^{\text{teacher lecture stability}} \Bigg)\end{aligned} \tag{5.38}$$

5.3.3 Model approximation

The basic MIP model can be of significant size, and extending it with the Δ-model can be demanding, sometimes doubling the model size or worse. Additionally, we would like to provide multiple perturbation suggestions as quickly as possible. Therefore, we introduce an approximation of the problem, which greatly reduces the model complexity by excluding events that are unlikely to have importance for the input events.

In the approximation, the model is built only including events that share persons or rooms with the input events. All other events are ignored except a few to avoid person double-booking problems.

The approximation model size grows with the number of distinct input persons, classes, and rooms. Therefore the approximation is less effective if the input includes events with many different entities. Such a scenario is rarely the case in practice, as the goal of this method is to provide targeted changes, and the input events are thus likely to focus on specific persons or classes.

5.4 Algorithm

Solving the Δ-model results in a perturbed timetable, where the best possible improvements are made for the input entities while considering consequences for the timetable as a whole. We extract the perturbation as a suggestion. To produce a second suggestion, we need to find the second best solution, the third suggestion corresponds to the third best solution, and so on. Thereby, we would like to find the k best solutions to the model. Two solution approaches for achieving this are considered. The first option is to iteratively solve the model and introduce a cut prohibiting the found solution until k solutions have been found. The second option is setting the MIP solver parameters to find the k best solutions, thereby continuing the optimization process after the optimal solution has been found. This benefits from finding the solutions (suggestions) in one optimization process, which is generally faster.

In our implementation, we use the first approach, as this allows us to use a free and open-source MIP solver in production, which does not support finding the k best solutions. We have also observed situations where the second approach "gets stuck" finding additional solutions or proving optimality, to such an extent that the first approach is faster. Additionally, we want to show the user suggestions immediately as they are found, which is significantly more straightforward with the iterative approach.

A suggestion $\mathcal{S}$ is extracted from a solution by simply looking at which events have moved to a new timeslot, i.e., $\mathcal{S} = \{\, (e,t) \in \mathcal{E} \times \mathcal{T} \mid y_{et} = 1 \land y^*_{et} = 0 \,\}$. To prohibit suggestion $\mathcal{S}$, the cut shown in (5.39) is added to the model before solving it again.

$$\sum_{(e,t)\in\mathcal{S}} y_{et} \leq |\mathcal{S}| - 1 \tag{5.39}$$

The iterative process of solving the model, extracting a suggestion, and applying a cut continues until one of multiple termination conditions is met. These conditions consider total runtime and suggestions count and the time required for a single model solve. Each possible termination condition is:

- Found the minimum required number of suggestions **and** has run the minimum required amount of time
- Found the maximum number of allowed suggestions
- Has run the maximum allowed amount of time

- A single solve took too long (terminated due to time limit)
- A suggestion is found that would unassign an event (moved to dummy-timeslot)

The combination of these termination conditions gives the best user experience. Meaning that if the model is easy to solve, the algorithm is allowed to find more suggestions – within a limit as not to overwhelm the user. Conversely, if the model is hard, it is given some time to find suggestions but stops if it takes too long. The last termination condition is to stop if a suggestion removes an event from the timetable. Unassigning an event is associated with a cost outweighing the Δ objectives, so such a solution is only found when no other feasible solutions exist. This is sometimes the case in tightly packed timetables when using the approximated model.

We show each suggestion immediately as it becomes available and includes an overview of its consequences. Thereby the timetabler can consider the suggestions while the algorithm continues to find more. At any time, the timetabler can choose a suggestion (updating the timetable) or terminate the search.

Algorithm 1 shows the implemented algorithm.

Algorithm 1 Iterative Δ-model

```
input: Timetable $T$, input events $\mathcal{E}_m$, bool useApproximation
$M$ = BuildMIP($T$, $\mathcal{E}_m$, useApproximation)
Fix $M$ assignment and solve                    ▷ Set initial variable values
Apply $\Delta$-model to $M$
loop
    Solve $M$
    Extract suggestion $\mathcal{S}$ from $M$          ▷ $\mathcal{S}$ returned to user immediately
    if Termination condition met then
        return
    end if
    Add cut to $M$
end loop
```

5.5 Results

The purpose of this section is to evaluate the implemented methods. To that end, we have chosen five different real-world datasets of varying sizes from the Lectio database. Table 5.3 provides an overview of these timetables showing the timetable entity set sizes.

We test the method with two different modes of choosing input events: using an EventChain (a single event or multiple chained together) or all events associated with a class. In both cases, we create a list of all possible inputs ordered by the number of events. Then 25 of these sets are picked uniformly from the ordered list to be used as input. Thereby we have guaranteed varied input ranging from smallest to largest possible number of input events.

Additionally, we test the method using both the full and approximated model. The right side of Table 5.3 shows an overview of the number of events included in the approximated model. This overview shows the average and the maximum number of events included for the 25 different inputs. Considering the maximum number of included events, the approximated model in the worst case includes between 12% and 41% of events (T4 and T1 single input respectively). A rather substantial reduction.

	Set sizes					Events in Approximation (average, max)	
Dataset	Events	Teachers	Students	Classes	Rooms	Single input	Class input
T1	388	43	561	193	71	(54.0, 158)	(60.4, 105)
T2	865	78	1.039	274	67	(88.5, 211)	(98.4, 165)
T3	776	46	450	290	68	(68.8, 107)	(67.7, 115)
T4	1.620	105	877	476	93	(89.4, 199)	(141.3, 237)
T5	1.952	119	1.405	509	96	(256.5, 358)	(318.5, 475)

Table 5.3: Overview of sets of interest for each timetable as well as the number of events included in the approximation.

Reducing the number of events included in the model has a significant effect on model sizes. Table 5.4 shows the number of variables and constraints of the original MIP model and the Δ-model for each method parameter setting. For the Δ-model, the shown values are the averages of the 25 inputs. The table shows that extending the MIP with the Δ-model in the worst case increases the number of constraints and variables by 136% and 49%, respectively. The approximation is very effective at reducing the sizes of the Δ-models. All have both fewer constraints and variables compared to the original non-extended model.

An important aspect of the method is to provide perturbation suggestions quickly to provide value rapidly and feel responsive to the user. The algorithm is run on all inputs, using all parameter settings, on all test timetables with the same termination condition parameters as are used in production:

- Min. suggestions / total time (seconds): 5 / 120
- Max. suggestions / total time (seconds): 20 / 500
- Max. single MIP solve time (seconds): 180

We perform all computational tests on a computer running Windows 10 64bit equipped with an Intel i7-6700K CPU clocked at 4.0 GHz and with 32GB of RAM. We use Gurobi 9.0 is used as the MIP solver with a single thread and default settings.

Figure 5.6 shows the average time per suggestion for all 25 inputs and truly shows the approximation's impact. For all timetables, the approximation produces suggestions much more quickly. Even for the two largest timetables T4

				% from Original	
Dataset	Model	Constraints	Variables	Constraints	Variables
T1	Original	28,652	25,282	-	-
	Δ Single Full	45,441	28,637	59%	13%
	Δ Class Full	45,418	28,654	59%	13%
	Δ Single Approx.	24,146	13,971	-16%	-45%
	Δ Class Approx.	25,370	14,509	-11%	-43%
T2	Original	99,276	86,790	-	-
	Δ Single Full	190,084	127,710	91%	47%
	Δ Class Full	189,968	127,693	91%	47%
	Δ Single Approx.	77,953	44,016	-21%	-49%
	Δ Class Approx.	85,182	47,372	-14%	-45%
T3	Original	47,376	49,953	-	-
	Δ Single Full	104,265	74,580	120%	49%
	Δ Class Full	104,211	74,583	120%	49%
	Δ Single Approx.	26,971	18,937	-43%	-62%
	Δ Class Approx.	27,304	19,077	-42%	-62%
T4	Original	142,225	157,492	-	-
	Δ Single Full	336,020	248,660	136%	58%
	Δ Class Full	335,875	248,652	136%	58%
	Δ Single Approx.	103,468	61,747	-27%	-61%
	Δ Class Approx.	135,448	76,915	-5%	-51%
T5	Original	325,622	168,021	-	-
	Δ Single Full	348,348	230,511	7%	37%
	Δ Class Full	348,206	230,525	7%	37%
	Δ Single Approx.	175,450	91,689	-46%	-45%
	Δ Class Approx.	189,712	98,344	-42%	-41%

Table 5.4: The model sizes for each timetable. Values for the Δ-model are averages of the models resulting from the 25 different inputs.

and T5, the approximated model produces suggestions faster than, or comparable to, the full model for the smallest timetable T1. Not all inputs result in the same number of suggestions and the average time per suggestion only includes those algorithm runs that produced that number of suggestions. This explains the downward dips in some of the plots, as some input results in models that are more difficult to solve, making it impossible to produce the same number of suggestions within the same amount of time. Thereby, data points corresponding to few suggestions include more runs while those for many suggestions generally include easier models.

Using “Single” or “Class” input does not significantly affect the difficulty of the model. Both types of input show roughly the same average time per suggestion. Including more input events does increase complexity by further opening the model, as additional events can be considered for objective improvements and perturbation starting points. However, as shown in Table 5.4, the model sizes are very similar. Only when using the approximated model does the "Class" input result in slightly larger models.

The approximation is only usable if it produces suggestions of high enough quality. To evaluate suggestion quality, we consider the relative gap from the cost of the perturbed timetable to the cost of the original. We calculate the relative gap by

$$(cost(T) - cost(T_{\mathcal{S}}))\,/cost(T_{\mathcal{S}}) \cdot 100\%$$

where T is the original timetable and $T_{\mathcal{S}}$ is the timetable perturbed by suggestion $\mathcal{S}$. Thereby a positive value corresponds to an improvement in solution value, and conversely, a negative value indicates a decrease. Figure 5.7 shows the average relative gap per suggestion for all timetables and parameter settings. This figure shows that the full model does produce better suggestions, generally finding improvements to the timetable. The largest difference in relative gap for the same input and model type is seen in timetable T2 for suggestion number 15 using the "Single" parameter. Here there is a difference of 0.98% between the approximated and full model. An average difference in solution quality of less than 1% is satisfactory.

The trend of the average solution quality for timetable T1 does behave slightly differently compared to the other timetables: the difference between the approximation and the full model is smaller, the trend of average solution quality for the full model is more erratic, and there are cases where the approximation outperforms the full model. Investigations reveal that the smallest timetable, T1, is quite difficult when it comes to providing perturbation suggestions. This timetable is very compact and has many events that have few feasible timeslot assignments. Thereby, a greater number of perturbations are required to provide valid suggestions, which is more difficult when using the full model, resulting in models that are sometimes not solvable within the given time limits. Additionally, there are cases where it is impossible to find suggestions because the input is already assigned to its only valid assignment. This is supported by Table 5.5 and 5.6 which shows the termination conditions and number of moves per suggestion respectively.
Out of 100 runs (25 runs with four different settings), 65 could not find any suggestions: 30 due to solver time limit and 35 because there were no valid suggestions. Only 22 of the 100 runs terminate normally, which is much less than any other timetable. Table 5.6 shows that timetable T1 requires more moves per suggestion, with 40.9% of suggestions proposing six or more moves. For the other timetables, the average percentage of suggestions with six moves or more is only 13.5%. Across all five timetables, 42.8% of found suggestions include four or more moves. Manually identifying perturbations including four or more events can be difficult, and therefore this method helps the timetabler to make timetable changes they have little chance of identifying themselves.

5.6 Conclusion

This paper proposed a novel approach for providing educational timetabling decision support by suggesting timetable perturbations. The proposed method

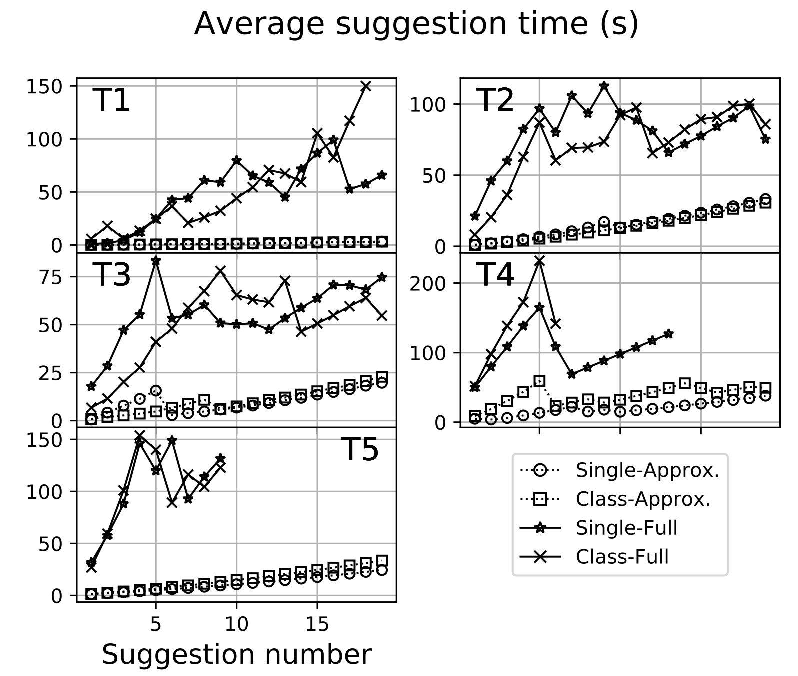


Figure 5.6: The average cumulative time in seconds to find each suggestion (at most 20) using every parameter setting on all timetables.

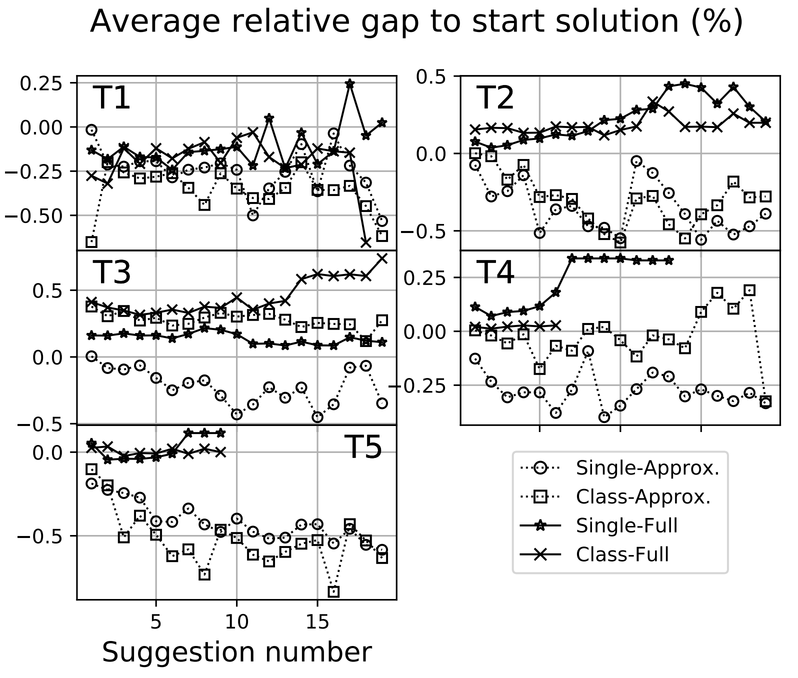


Figure 5.7: The average gap (%) to the starting solution for each suggestion (at most 20) using every parameter setting on all timetables.

	Termination condition				
	Normal	TimeLimit		NoMoreValid	
Dataset		+	-	+	-
T1	22	4	30	9	35
T2	91	8	1	0	0
T3	93	4	3	0	0
T4	64	19	17	0	0
T5	57	24	16	0	3

Table 5.5: The termination conditions for 100 runs using the four parameter settings on each timetable. The TimeLimit condition represent a run that is terminated because a single MIP solve reached the time limit and NoMoreValid when no more feasible solutions exist. '+' shows that feasible solutions/suggestions are found before termination (and '-' when no solutions). Normal termination is when the run is finished normally given the total number of suggestions and total algorithm time limits.

	Number of moves							
Dataset	1	2	3	4	5	6-10	11-15	16-20
T1	0	30	33	58	80	135	2	2
T2	257	486	354	161	81	30	0	0
T3	151	170	253	330	118	368	43	3
T4	259	266	217	120	53	69	8	0
T5	76	167	252	244	157	151	11	0
%	14.3	21.5	21.3	17.6	9.4	14.5	1.2	0.1

Table 5.6: The number of suggestions which include a given number of moves for all timetables using the four parameter settings.

is a Mixed Integer Programming model-based approach, and we have shown how to apply it to the Timetabling Problem at Danish high schools. This should provide inspiration for how to apply this approach to other timetabling problems and model definitions.

We evaluated the method with varied use cases on five different real-world timetables. The results show that the method can provide perturbation suggestions that are difficult for a practical timetabler to identify manually. While the full model does provide suggestions of higher quality, the approximated model is much quicker and results in suggestions that are close in quality. The reason is that the approximation significantly reduced the MIP model size while including the essential elements of the problem. Using the method, we can quickly provide the timetabler with valuable suggestions, ensuring a positive user experience.

Two immediate possible agendas for future work are to include rooms and to allow the timetabler more control. It would be interesting to look into the value of amending the method to consider room changes and how to do so without significantly increasing the complexity. In the presented work, we focus on all students, teachers, and classes in the given input, but it could be beneficial to allow the timetabler to specify the method's focus, either specific quality metrics or individual persons or classes. Especially the latter research suggestion would increase the practical value of this already beneficial method.

Acknowledgements

Rasmus Ø. Mikkelsen's industrial Ph.D. project is funded by Innovation Fund Denmark (IFD). IFD has supported the work solely financially and has not participated in any research-related activities.

References

Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. (2015). "An overview of curriculum-based course timetabling". In: *Top* 23.2, pp. 313–349. DOI: 10.1007/s11750-015-0363-2. URL: http://dx.doi.org/10.1007/s11750-015-0366-z.

Cacchiani, V., Caprara, A., Roberti, R., and Toth, P. (2013). "A new lower bound for curriculum-based course timetabling". In: *Computers and Operations Research* 40.10, pp. 2466–2477. ISSN: 03050548. DOI: 10.1016/j.cor.2013.02.010. URL: http://dx.doi.org/10.1016/j.cor.2013.02.010.

El Sakkout, H. and Wallace, M. (2000). "Probe backtrack search for minimal perturbation in dynamic scheduling". In: *Constraints* 5.4, pp. 359–388.

Junginger, W. (1986). "Timetabling in Germany—a survey". In: *Interfaces* 16.4, pp. 66–74.

Lewis, R. (2008). "A survey of metaheuristic-based techniques for university timetabling problems". In: *OR spectrum* 30.1, pp. 167–190.

Lindahl, M., Mason, A. J., Stidsen, T., and Sørensen, M. (2018). "A strategic view of University timetabling". In: *European Journal of Operational Research* 266.1, pp. 35–45.

Lindahl, M., Stidsen, T., and Sørensen, M. (2019). "Quality recovering of university timetables". In: *European Journal of Operational Research* 276.2, pp. 422–435.

Schaerf, A. (1999). "A survey of automated timetabling". In: *Artificial intelligence review* 13.2, pp. 87–127.

Sørensen, M. and Stidsen, T. R. (2013). *Comparing Solution Approaches for a Complete Model of High School Timetabling.* Tech. rep. March.

Chapter 6

A MIP Formulation of the International Timetabling Competition 2019 Problem

Dennis Søren Holm[a] · Rasmus Ørnstrup Mikkelsen[a,b] · Matias Sørensen[b] · Thomas Jacob Riis Stidsen[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]MaCom A/S, Vesterbrogade 48, 1., DK-1620 København V, Denmark

Status: Published as a Technical Report

Keywords: Mixed Integer Programming · University Timetabling · International Timetabling Competition 2019 · ITC 2019

6.1 Introduction

This report considers the problem presented in the International Timetabling Competition 2019 (ITC 2019). The ITC 2019 problem description presents a generalized model for the University Course Timetabling problem combined with an XML data format. The generalized model aims to include most of the aspects that universities worldwide might consider when constructing a timetable. The model has been simplified to some extent, but the complexity of the problem remains. To understand the origin of the data, the data format, and to get a more in-depth description of the generalized model, it is encouraged to read the ITC 2019 problem description (Müller et al., 2018).

When considering an operational research problem like the ITC 2019, it can be beneficial to describe the problem with a mathematical model. It is an excellent way to check if the problem formulation has been wholly understood. It also provides some idea of the problem's complexity. Additionally, if a commercial solver can solve the mathematical model, no further work needs to be done. Moreover, a mathematical model can serve as a basis for developing matheuristics and can also validate solutions from other solution methods, i.e., metaheuristics.

In this report, the ITC 2019 problem is described using a linear Mixed Integer Programming (MIP) model. The MIP model has been verified by comparing objective penalties and feasibility of solutions with the validator provided for the ITC 2019 (www.itc2019.org/validator).

6.2 Notations

This section contains an overview of the notation for the MIP model. Table 6.1 shows the sets that are used. Table 6.2 contains the notation of the parameters that are given in the XML data. Table 6.3 shows other modelling notation.

Symbol	Description
Δ	Set of distribution constraints
$\mathcal{P}$	Set of penalty variables
T	Set of time slots
$\mathcal{D}$	Set of days
$\mathcal{W}$	Set of weeks
$\mathcal{K}$	Set of courses
$\mathcal{C}$	Set of classes
$\mathcal{T}$	Set of times
$\mathcal{R}$	Set of rooms
$\mathcal{S}$	Set of students
$\mathcal{C}_\delta$	Set of classes for a distribution constraint $\delta \in \Delta$
$\mathcal{C}_s$	Set of classes a student $s \in \mathcal{S}$ can attend
$\mathcal{C}_\zeta$	Set of classes of a subpart $\zeta \in \mathrm{Z}_\omega$
$\mathcal{R}_c$	Set of available rooms for a class $c \in \mathcal{C}$
$\mathcal{T}_c$	Set of available times for a class $c \in \mathcal{C}$
$\mathcal{K}_s$	Set of courses a student $s \in \mathcal{S}$ must attend
$\mathcal{S}_c$	Set of students that can attend a class $c \in \mathcal{C}$
$\mathcal{S}_k$	Set of students that must attend a course $k \in \mathcal{K}$
Ω_k	Set of configurations of a course $k \in \mathcal{K}$
Z_ω	Set of subpart of a configuration $\omega \in \Omega_k$

Table 6.1: Set notations.

Symbol	Description
c_δ	Cost of the distribution constraint $\delta \in \Delta$
$p_{c,t}$	Penalty of assigning time $t \in \mathcal{T}$ to class $c \in \mathcal{C}$
$p_{c,r}$	Penalty of assigning room $r \in \mathcal{R}$ to class $c \in \mathcal{C}$
ψ_t	Objective weight of the time penalties
ψ_r	Objective weight of the room penalties
ψ_δ	Objective weight of the distribution constraint penalties
ψ_s	Objective weight of the student conflict penalties
$\mathtt{D}$	Distribution constraint parameter
$\mathtt{G}$	Distribution constraint parameter
$\mathtt{M}$	Distribution constraint parameter
$\mathtt{R}$	Distribution constraint parameter
$\mathtt{S}$	Distribution constraint parameter
t^{start}	The starting time slot of time $t \in \mathcal{T}$, $t^{\text{start}} \in \mathrm{T}$
t^{length}	The duration in time slots of time $t \in \mathcal{T}$
t^{end}	The ending time of time $t \in \mathcal{T}$, $t^{\text{start}} + t^{\text{length}} = t^{\text{end}} \in \mathrm{T}$
t^{days}	The set of days of time $t \in \mathcal{T}$, $t^{\text{days}} \subseteq \mathcal{D}$
$t^{\text{days.first}}$	The first day of time $t \in \mathcal{T}$, $t^{\text{days.first}} \in \mathcal{D}$
t^{weeks}	The set of weeks of time $t \in \mathcal{T}$, $t^{\text{weeks}} \subseteq \mathcal{D}$
$t^{\text{weeks.first}}$	The first week of time $t \in \mathcal{T}$, $t^{\text{weeks.first}} \in \mathcal{W}$

Table 6.2: Parameter notations. Parameters are given in the data sets.

Symbol	Description
c_i	A specific class with ID $i \in \mathbb{Z}^+$
c_i^{parent}	The parent class of class c_i, $c_i^{\text{parent}} \in \mathcal{C}$
c^{limit}	The student limit of class c
$\tilde{r}$	A 'dummy' room, which only exists in the model and does not follow the rules of a regular room.
r_i	A room of class c_i, $r_i \in \mathcal{R}_{c_i}$
t_i	A time of class c_i, $t_i \in \mathcal{T}_{c_i}$
$\bar{t}$	Another time different from $t \in \mathcal{T}$, $\bar{t} \in \mathcal{T}$
$\bar{\tau}$	Another time slot different from $\tau \in \mathrm{T}$, $\bar{\tau} \in \mathrm{T}$
c_p	Cost of the penalty $p \in \mathcal{P}$
T'	Set of start time slots, $\mathrm{T}' \subseteq \mathrm{T}$
T''	Set of end time slots, $\mathrm{T}'' \subseteq \mathrm{T}$
M	Big-M

Table 6.3: Other modelling notation.

6.3 Decision variables

The model includes two main decision variables; the scheduling variable $x_{c,t,r}$ and the student sectioning variable $e_{s,c}$. The scheduling variable is binary with indices, classes, times, and rooms. It is defined as

$$x_{c,t,r} = \begin{cases} 1 & \text{if class } c \in \mathcal{C} \text{ is scheduled in time } t \in \mathcal{T}_c \text{ in room } r \in \mathcal{R}_c \\ 0 & \text{otherwise} \end{cases}$$

and is defined for all $c \in \mathcal{C}$, $t \in \mathcal{T}_c$ and $r \in \mathcal{R}_c$. If the class does not need to be assigned a room we set $\mathcal{R}_c = \{\tilde{r}\}$, where $\tilde{r}$ is a 'dummy' room and $\tilde{r} \notin \mathcal{R}$.
The scheduling variables $x_{c,t,r}$ (class-time-room) leads to the following auxiliary variables $y_{c,t}$ (class-time), $z_{c,d}$ (class-day) and $w_{c,r}$ (class-room).

$$y_{c,t} = \begin{cases} 1 & \text{if class } c \in \mathcal{C} \text{ is scheduled in time } t \in \mathcal{T}_c \\ 0 & \text{otherwise} \end{cases}$$

$$z_{c,d} = \begin{cases} 1 & \text{if class } c \in \mathcal{C} \text{ is scheduled on day } d \\ 0 & \text{otherwise} \end{cases}$$

$$w_{c,r} = \begin{cases} 1 & \text{if class } c \in \mathcal{C} \text{ is scheduled in room } r \in \mathcal{R}_c \\ 0 & \text{otherwise} \end{cases}$$

The student sectioning variable $e_{s,c}$ is also binary with indices; students and classes. It is defined as

$$e_{s,c} = \begin{cases} 1 & \text{if student } s \in \mathcal{S} \text{ is attending class } c \in \mathcal{C}_s \\ 0 & \text{otherwise} \end{cases}$$

and is defined for all $s \in \mathcal{S}$ and $c \in \mathcal{C}_s$.

6.4 Constraints

The constraints are divided into three categories; the primary constraints that ensure general timetable feasibility, the distribution constraints that define the feasibility between the scheduling of classes, and the student sectioning constraints that ensure that the students are assigned the classes according to the enrollment and the structure of the courses.

6.4.1 Primary constraints

The primary constraints ensure that all classes are scheduled and that no room can be double booked.

$$\sum_{t \in \mathcal{T}_c} \sum_{r \in \mathcal{R}_c} x_{c,t,r} = 1 \qquad \forall\, c \in \mathcal{C}$$

This constraint defines that all classes must be assigned a time and a room (possibly the dummy-room if no room is required).

$$\sum_{\substack{c \in \mathcal{C}, \\ \bar{t} \in \mathcal{T}: \\ t.\texttt{Overlap}(\bar{t})}} x_{c,\bar{t},r} + M \sum_{c \in \mathcal{C}} x_{c,t,r} \leq M \qquad \forall\, r \in \mathcal{R}, t \in \mathcal{T}$$

The constraint ensures that if a class is scheduled in room r in time t, then there can be no classes scheduled in any overlapping times in the same room. The `Overlap` function is described further in appendix 6.A. The big-M is equal to the number of classes $M = |\mathcal{C}|$.

Furthermore we have the following constraints to control the auxiliary variables

$$\sum_{r \in \mathcal{R}_c} x_{c,t,r} = y_{c,t} \qquad \forall\, c \in \mathcal{C}, t \in \mathcal{T}_c$$

$$\sum_{\substack{t \in \mathcal{T}_c : d \in t^{\text{days}}, \\ r \in \mathcal{R}_c}} x_{c,t,r} = z_{c,d} \qquad \forall\, c \in \mathcal{C}, d \in \mathcal{D}$$

$$\sum_{t \in \mathcal{T}_c} x_{c,t,r} = w_{c,r} \qquad \forall\, c \in \mathcal{C}, r \in \mathcal{R}_c$$

6.4.2 Distribution constraints

In this section, the modeling of the distribution constraints of different types is described. Each subsection considers a single distribution constraint of the given type. The constraints presented are therefore generated for each distribution constraint of that type. If any auxiliary variables are defined, they have an extra dimension δ that is not written explicitly, such that auxiliary variables from two different distribution constraints are not mixed. Each type of distribution constraint can be either soft or hard. Each soft constraint will include a penalty variable p, which dimensions will not be stated explicitly. The cost of the penalty variable is denoted by c_p. Each distribution constraint has a set of classes $\mathcal{C}_\delta$. The soft distribution constraints have a penalty c_δ. It will be stated in the section if the distribution constraint has parameters.

6.4.2.1 SameStart

The `SameStart` constraint says that if a class c_i is assigned time t_i, which starts at time slot τ, then c_j cannot be assigned a time that starts at a different time slot. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad \sum_{\substack{t_i \in \mathcal{T}_{c_i}: \\ t_i^{\text{start}} = \tau}} y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{start}} \neq \tau}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, \tau \in \bigcup_{t \in \mathcal{T}_{c_i}} t^{\text{start}}$$

$$\text{Soft:} \quad \sum_{\substack{t_i \in \mathcal{T}_{c_i}: \\ t_i^{\text{start}} = \tau}} y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{start}} \neq \tau}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, \tau \in \bigcup_{t \in \mathcal{T}_{c_i}} t^{\text{start}}$$

6.4.2.2 SameTime

The `SameTime` constraint says that if a class c_i is assigned time t_i, then c_j cannot be assigned a time that is not taught at the same time. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{start}} \leq t_j^{\text{start}} \wedge t_j^{\text{end}} \leq t_i^{\text{end}}) \\ \vee (t_j^{\text{start}} \leq t_i^{\text{start}} \wedge t_i^{\text{end}} \leq t_j^{\text{end}}))}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{start}} \leq t_j^{\text{start}} \wedge t_j^{\text{end}} \leq t_i^{\text{end}}) \\ \vee (t_j^{\text{start}} \leq t_i^{\text{start}} \wedge t_i^{\text{end}} \leq t_j^{\text{end}}))}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.3 DifferentTime

The `DifferentTime` constraint says that if a class c_i is assigned time t_i, then c_j cannot be assigned a time that overlaps the same time of the day. Here we have $p \in \{0, 1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{end}} \le t_j^{\text{start}}) \\ \vee (t_j^{\text{end}} \le t_i^{\text{start}}))}} y_{c_j,t_j} \le 1 \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{end}} \le t_j^{\text{start}}) \\ \vee (t_j^{\text{end}} \le t_i^{\text{start}}))}} y_{c_j,t_j} - 1 \le p \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.4 SameDays

The `SameDays` constraint says that if a class c_i is assigned time t_i, with day-set t_i^{days}, then c_j cannot be assigned a time with day-set t_j^{days} if the smaller of the day-sets is not included in the larger. Here we have $p \in \{0, 1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \not\subseteq t_j^{\text{days}} \wedge \\ t_j^{\text{days}} \not\subseteq t_i^{\text{days}}}} y_{c_j,t_j} \le 1 \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \not\subseteq t_j^{\text{days}} \wedge \\ t_j^{\text{days}} \not\subseteq t_i^{\text{days}}}} y_{c_j,t_j} - 1 \le p \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.5 DifferentDays

The `DifferentDays` constraint says that if a class c_i is assigned time t_i, with day-set t_i^{days}, then c_j cannot be assigned a time with day-set t_j^{days} if the two sets have any days in common. Here we have $p \in \{0, 1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset}} y_{c_j,t_j} \le 1 \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset}} y_{c_j,t_j} - 1 \le p \quad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.6 SameWeeks

The `SameWeeks` constraint says that if a class c_i is assigned time t_i, with week-set t_i^{weeks}, then c_j cannot be assigned a time with week-set t_j^{weeks} if the smaller of the week-sets is not included in the larger. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \not\subseteq t_j^{\text{weeks}} \wedge \\ t_j^{\text{weeks}} \not\subseteq t_i^{\text{weeks}}}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \not\subseteq t_j^{\text{weeks}} \wedge \\ t_j^{\text{weeks}} \not\subseteq t_i^{\text{weeks}}}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.7 DifferentWeeks

The `DifferentWeeks` constraint says that if a class c_i is assigned time t_i, with week-set t_i^{weeks}, then c_j cannot be assigned a time with day-set t_j^{weeks} if the two sets have any weeks in common. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.8 Overlap

The `Overlap` constraint says that if a class c_i is assigned time t_i, then c_j cannot be assigned a time that does **not** overlap t_i. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_j^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset))}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_j^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset))}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.9 NotOverlap

The `NotOverlap` constraint says that if a class c_i is assigned time t_i, then c_j cannot be assigned a time that overlaps t_i. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ (t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset)}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \qquad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ ((t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset)}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.10 SameRoom

The `SameRoom` constraint says that if a class c_i is assigned room r_i, then another class c_j cannot be assigned another room. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad w_{c_i,r_i} + \sum_{r_j \in \mathcal{R}_{c_j} \setminus \{r_i\}} w_{c_j,r_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, r_i \in \mathcal{R}_{c_i}$$

$$\text{Soft:} \qquad w_{c_i,r_i} + \sum_{r_j \in \mathcal{R}_{c_j} \setminus \{r_i\}} w_{c_j,r_j} - 1 \leq p \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j \in \mathcal{C}_\delta, r_i \in \mathcal{R}_{c_i}$$

6.4.2.11 DifferentRoom

The `DifferentRoom` constraint says that if a class c_i is assigned room r_i, then another class c_j cannot be assigned the same room. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad w_{c_i,r} + w_{c_j,r} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, c_j \in \mathcal{C}_\delta, r \in \mathcal{R}_{c_i} \cup \mathcal{R}_{c_j}$$

$$\text{Soft:} \qquad w_{c_i,r} + w_{c_j,r} - 1 \leq p \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, c_j \in \mathcal{C}_\delta, r \in \mathcal{R}_{c_i} \cup \mathcal{R}_{c_j}$$

6.4.2.12 SameAttendees

The `SameAttendees` constraint says that if a class c_i is scheduled at time t_i in room r_i, then another class c_j cannot be scheduled such that the times overlap (like the `Overlap` constraint) but also not such that the the classes overlap in a time-room sense. That means that the classes must be scheduled such that the travel time between the two assigned rooms does not exceed the duration between the two assigned times. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard: } x_{c_i,t_i,r_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i\texttt{.Overlap}(t_j)}} y_{c_j,t_j} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg t_i\texttt{.Overlap}(t_j), \\ r_j \in \mathcal{R}_{c_j}: \\ t_i\texttt{.Overlap}(t_j, r_j, r)}} x_{c_j,t_j,r_j} \leq 1 \; \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in t_{c_i}, r_i \in \mathcal{R}_{c_i}$$

$$\text{Soft: } x_{c_i,t_i,r_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i\texttt{.Overlap}(t_j)}} y_{c_j,t_j} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg t_i\texttt{.Overlap}(t_j), \\ r_j \in \mathcal{R}_{c_j}: \\ t_i\texttt{.Overlap}(t_j, r_j, r)}} x_{c_j,t_j,r_j} - 1 \leq p \; \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in t_{c_i}, r_i \in \mathcal{R}_{c_i}$$

6.4.2.13 Precedence

The `Precedence` constraint says that if a class c_i is assigned time t_i, then another class c_j cannot be assigned a time that starts in an earlier week or on an earlier day of the week (if they start in the same week) or on an earlier time (if they start in the same week on the same day). Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard: } \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{weeks.first}} < t_i^{\text{weeks.first}} \vee \\ (t_j^{\text{weeks.first}} = t_i^{\text{weeks.first}} \wedge \\ (t_j^{\text{days.first}} < t_i^{\text{days.first}} \vee \\ (t_j^{\text{days.first}} = t_i^{\text{days.first}} \wedge t_j^{\text{start}} < t_i^{\text{start}})))}} y_{c_j,t_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft: } \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{weeks.first}} < t_i^{\text{weeks.first}} \vee \\ (t_j^{\text{weeks.first}} = t_i^{\text{weeks.first}} \wedge \\ (t_j^{\text{days.first}} < t_i^{\text{days.first}} \vee \\ (t_j^{\text{days.first}} = t_i^{\text{days.first}} \wedge t_j^{\text{start}} < t_i^{\text{start}})))}} y_{c_j,t_j} - 1 \leq p \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.14 WorkDay(S)

The `WorkDay(S)` constraint says that if a class c_i is assigned time t_i, then another class c_j cannot be assigned a time that overlaps any week and any day such that the time difference between earliest start time and latest end time is greater than the parameter `S`. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset \, \wedge \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset \, \wedge \\ \max(t_i^{\text{end}}, t_j^{\text{end}}) - \min(t_i^{\text{start}}, t_j^{\text{start}}) > \mathtt{S}}} y_{c_j,t_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset \, \wedge \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset \, \wedge \\ \max(t_i^{\text{end}}, t_j^{\text{end}}) - \min(t_i^{\text{start}}, t_j^{\text{start}}) > \mathtt{S}}} y_{c_j,t_j} - 1 \leq p \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.15 MinGap(G)

The `MinGap(G)` constraint says that if a class c_i is assigned time t_i, then another class c_j cannot be assigned a time that overlaps any week and any day such that the time between the earliest end time and the latest start time is less than `G`. Here we have $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg(t_i^{\text{weeks}} \cap t_i^{\text{weeks}} = \emptyset \, \vee \\ t_i^{\text{days}} \cap t_i^{\text{days}} = \emptyset \, \vee \\ t_i^{\text{end}} + \mathtt{G} \leq t_j^{\text{start}} \, \vee \\ t_j^{\text{end}} + \mathtt{G} \leq t_i^{\text{start}})}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

$$\text{Soft:} \quad y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg(t_i^{\text{weeks}} \cap t_i^{\text{weeks}} = \emptyset \, \vee \\ t_i^{\text{days}} \cap t_i^{\text{days}} = \emptyset \, \vee \\ t_i^{\text{end}} + \mathtt{G} \leq t_j^{\text{start}} \, \vee \\ t_j^{\text{end}} + \mathtt{G} \leq t_i^{\text{start}})}} y_{c_j,t_j} - 1 \leq p \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6.4.2.16 MaxDays(D)

The `MaxDays(D)` constraint says that the given classes cannot be spread over more than `D` days. We define the auxiliary variable:

$$\gamma_d = \begin{cases} 1 & \text{if any class } c \in \mathcal{C}_\delta \text{ is scheduled on day } d \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases}$$

For all days $d \in \mathcal{D}$ The variable γ_d is bounded by the constraint:

$$\sum_{c \in \mathcal{C}_\delta} z_{c,d} \leq M\gamma_d \qquad \forall d \in \mathcal{D}$$

Where $M = |\mathcal{C}_\delta|$. The `MaxDays(D)` constraints are shown below, where $p \in \mathbb{Z}^+$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad \sum_{d \in \mathcal{D}} \gamma_d \leq \mathtt{D}$$

$$\text{Soft:} \qquad \sum_{d \in \mathcal{D}} \gamma_d - \mathtt{D} \leq p$$

6.4.2.17 MaxDayload(S)

The `MaxDayload(S)` says that the given classes cannot be scheduled such that the number of time slots on any day (day load) does not exceed `S`. We define the day load $\phi_{w,d} \in \mathbb{Z}^+$ on a day d in a week w.

$$\phi_{w,d} = \sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c : t^{week} = w \wedge t^{day} = d}} t^{\text{length}} y_{c,t} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

Thus the constraints are as follows.

$$\text{Hard:} \qquad \phi_{w,d} \leq \mathtt{S} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

$$\text{Soft:} \qquad \phi_{w,d} - \mathtt{S} \leq \iota_{w,d} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

The variable $\iota_{w,d} \in \mathbb{Z}^+$ counts the number of exceeding time slots on day d in week w. Thus the penalty for this distribution constraint is set by:

$$\frac{c_\delta}{|\mathcal{W}|} \sum_{\substack{w \in \mathcal{W}, \\ d \in \mathcal{D}}} \iota_{w,d} - 0.999 \leq p$$

The penalty should be computed using integer division. Since the division by $|\mathcal{W}|$ can result in non-integer value we subtract 0.999 to bind p correctly. As the distribution constraints cost c_δ is included in the above constraint, we have $p \in \mathbb{Z}^+$ and $c_p = 1$

6.4.2.18 MaxBreaks(R,S)

The `MaxBreaks(R,S)` constraint says that there can be no more than `R` breaks during a day, on any day in any week. A break is defined by having more than `S` empty timeslots between two consecutive classes. Two consecutive classes are considered to be in the same block if there is no break between them. This means that on any day d in any week w, the number of blocks $\beta_{w,d} \in \mathbb{Z}^+$ must be less than $\mathtt{R} + 1$.

$$\text{Hard:} \qquad \beta_{w,d} - 1 \leq \mathtt{R} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

$$\text{Soft:} \qquad \beta_{w,d} - 1 - \mathtt{R} \leq \eta_{w,d} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}$$

The variable $\eta_{w,d} \in \mathbb{Z}^+$ counts the number of exceeding time slots on day d in week w. Thus the penalty for this distribution constraint is set by:

$$\frac{c_\delta}{|\mathcal{W}|} \sum_{\substack{w \in \mathcal{W},\\ d \in \mathcal{D}}} \eta_{w,d} - 0.999 \leq p$$

Since the distribution constraint cost c_δ is included in the above constraint, we have $p \in \mathbb{Z}^+$ and $c_p = 1$

Constraints to control $\beta_{w,d}$

To count the number of blocks on a day d in a week w, we define the variable $\alpha_{w,d,\tau} \in \{0,1\}$. Let $\mathrm{T}' \subseteq \mathrm{T}$ be the set of time slots where any $c \in \mathcal{C}_\delta$ can start.

$$\alpha_{w,d,\tau} = \begin{cases} 1 & \substack{\text{if a block starts in week } w \in \mathcal{W} \text{ on day } d \\ \text{at time slot } \tau \in \mathrm{T}'} \\ 0 & \text{otherwise} \end{cases}$$

and thus we have

$$\beta_{w,d} = \sum_{\tau \in \mathrm{T}'} \alpha_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}$$

To set $\alpha_{w,d,\tau}$ correctly, we need the auxiliary variable

$$\sigma_{w,d,\tau} = \begin{cases} 1 & \substack{\text{if any class } c \in \mathcal{C}_\delta \text{ starts in week } w \in \mathcal{W} \\ \text{on day } d \text{ at time slot } \tau \in \mathrm{T}'} \\ 0 & \text{otherwise} \end{cases}$$

Which is controlled by the constraints:

$$\sum_{\substack{c \in \mathcal{C}_\delta,\\ t \in \mathcal{T}_c : t^{\text{start}} = \tau}} y_{c,t} \geq \sigma_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}'$$

$$\sum_{\substack{c \in \mathcal{C}_\delta,\\ t \in \mathcal{T}_c : t^{\text{start}} = \tau}} y_{c,t} \leq M \sigma_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}'$$

Where $M = |\mathcal{C}_\delta|$. We define another auxiliary variable

$$\varepsilon_{w,d,\tau} = \begin{cases} 1 & \substack{\text{if any class } c \in \mathcal{C}_\delta \text{ is scheduled in week } w \in \mathcal{W} \\ \text{on day } d \in \mathcal{D} \text{ and overlaps any time slot } \{\tau-1-\mathtt{S},\ldots,\tau-1\}} \\ 0 & \text{otherwise} \end{cases}$$

Which is controlled by the constraints:

$$\sum_{\substack{c \in \mathcal{C}_\delta,\\ t \in \mathcal{T}_c :\\ t.\mathtt{Overlap}(\{\tau-1-\mathtt{S},\ldots,\tau-1\})}} y_{c,t} \qquad \geq \varepsilon_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}$$

$$\sum_{\substack{c\in\mathcal{C}_\delta,\\ t\in\mathcal{T}_c:\\ t.\texttt{Overlap}(\{\tau-1-\texttt{S},\ldots,\tau-1\})}} y_{c,t} \qquad \leq M\varepsilon_{w,d,\tau} \qquad \forall\, w\in\mathcal{W}, d\in\mathcal{D}, \tau\in\mathrm{T}$$

Where $M = |\mathcal{C}_\delta|$. The auxiliary variables $\sigma_{w,d,\tau}$ and $\varepsilon_{w,d,\tau}$ sets $\alpha_{w,d,\tau}$ using the following constraints.

$$\sigma_{w,d,\tau} - \varepsilon_{w,d,\tau} \leq \alpha_{w,d,\tau} \qquad \forall\, w\in\mathcal{W}, d\in\mathcal{D}, \tau\in\mathrm{T}'$$

$$\sigma_{w,d,\tau} \geq \alpha_{w,d,\tau} \qquad \forall\, w\in\mathcal{W}, d\in\mathcal{D}, \tau\in\mathrm{T}'$$

$$\varepsilon_{w,d,\tau} + \alpha_{w,d,\tau} \leq 1 \qquad \forall\, w\in\mathcal{W}, d\in\mathcal{D}, \tau\in\mathrm{T}'$$

Note that for a time slot $\bar\tau$ where $\varepsilon_{w,d,\bar\tau} = 0$ because $\sum_{\substack{c\in\mathcal{C}_\delta,\\ t\in\mathcal{T}_c:\\ t.\texttt{Overlap}(\{\tau-1-\texttt{S},\ldots,\tau-1\})}} y_{c,t}$ is fixed to 0 (there exists no $y_{c,t}$ satisfying the sum conditions), then

$$\sigma_{w,d,\bar\tau} = \alpha_{w,d,\bar\tau} \qquad \forall\, w\in\mathcal{W}, d\in\mathcal{D}, \bar\tau\in\mathrm{T} : \left| \bigcap_{\substack{c\in\mathcal{C}_\delta,\\ t\in\mathcal{T}_c:\\ t.\texttt{Overlap}(\{\bar\tau-1-\texttt{S},\ldots,\bar\tau-1\})}} \{y_{c,t}\} \right| = 0$$

The `Overlap`-function tells if a time t overlaps a time period, similarly to the `Overlap` distribution constraint.

6.4.2.19 MaxBlock(M,S)

The `MaxBlock(M,S)` says that a block on any day in any week can be no longer than `M` time slots. Two consecutive classes are said to be in the same block if there are no more than `S` time slots between them. There exist special cases where a class by itself is longer than the maximum allowed time slots `M`. It is stated that such a class cannot be in a block with another class (or if the constraint is soft, we penalize when that happens). For the following constraints, we ignore the classes with length strictly larger than `M` (which is covered later). We define the variable $\rho_{w,d,\tau} \in \{0,1\}$

$$\rho_{w,d,\tau} = \begin{cases} 1 & \substack{\text{if a block longer than } \texttt{M} \text{ starts in week } w\in\mathcal{W} \\ \text{on day } d\in\mathcal{D} \text{ at time slot } \tau\in\mathrm{T}'} \\ 0 & \text{otherwise} \end{cases}$$

Then we can define the constraints

$$\text{Hard:} \qquad \sum_{\substack{w\in\mathcal{W},\\ d\in\mathcal{D},\\ \tau\in\mathrm{T}}} \rho_{w,d,\tau} = 0$$

$$\text{Soft:} \qquad \frac{c_\delta}{|\mathcal{W}|} \sum_{\substack{w\in\mathcal{W},\\ d\in\mathcal{D},\\ \tau\in\mathrm{T}}} \rho_{w,d,\tau} \leq p$$

Since the distribution constraints cost c_δ is included in the above constraint, we have $p \in \mathbb{Z}^+$ and $c_p = 1$

Constraints to control $\rho_{w,d,\tau}$

From section 6.4.2.18 we have defined $\alpha_{w,d,\tau}$ (any block starting at w, d, τ). Here we also need variable $\gamma_{w,d,\tau}$ defined as

$$\gamma_{w,d,\tau} = \begin{cases} 1 & \text{if a block ends in week } w \in \mathcal{W} \text{ on day } d \in \mathcal{D} \\ & \text{at time slot } \tau{\in}\mathrm{T}'' \\ 0 & \text{otherwise} \end{cases}$$

We have that

$$\rho_{w,d,\tau} \leq \alpha_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}$$

$$\mathtt{M}\rho_{w,d,\tau} + \mathtt{S} \sum_{\bar\tau \in \mathrm{T}: \tau < \bar\tau \leq \tau + \mathtt{M}} \gamma_{w,d,\bar\tau} \leq \mathtt{M} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}$$

$$\alpha_{w,d,\tau} - \sum_{\bar\tau \in \mathrm{T}: \tau < \bar\tau \leq \tau + \mathtt{M}} \gamma_{w,d,\bar\tau} \leq \rho_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}$$

To control $\gamma_{w,d,\tau}$, we need two auxiliary variables. Let $\mathrm{T}'' \subseteq \mathrm{T}$ be the set of time slots where any $c \in \mathcal{C}_\delta$ can end.

$$\varphi_{w,d,\tau} = \begin{cases} 1 & \text{if any class } c \in \mathcal{C}_\delta \text{ ends in week } w \in \mathcal{W} \text{ on day } d \in \mathcal{D} \\ & \text{at time slot } \tau{\in}\mathrm{T}'' \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t^{\text{end}} = \tau}} y_{c,t} \geq \varphi_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$

$$\sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t^{\text{end}} = \tau}} y_{c,t} \leq M\varphi_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$

Where $M = |\mathcal{C}_\delta|$. We define the variable $\theta_{w,d,\tau}$

$$\theta_{w,d,\tau} = \begin{cases} 1 & \text{if a class } c \in \mathcal{C}_\delta \text{ is scheduled in week } w{\in}\mathcal{W} \\ & \text{on day } d{\in}\mathcal{D} \text{ and overlaps } \{\tau{+}1,\ldots,\tau{+}1{+}\mathtt{S}\} \\ 0 & \text{otherwise} \end{cases}$$

and the constraints

$$\sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t.\mathtt{Overlaps}(\tau+1,\ldots,\tau+1+\mathtt{S})}} y_{c,t} \geq \theta_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$

$$\sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t.\mathtt{Overlaps}(\tau+1,\ldots,\tau+1+\mathtt{S})}} y_{c,t} \leq M\theta_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$

We can now define the constraints on $\gamma_{w,d,\tau}$.

$$\varphi_{w,d,\tau} - \theta_{w,d,\tau} \leq \gamma_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$
$$\varphi_{w,d,\tau} \geq \gamma_{w,d,\tau} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$
$$\theta_{w,d,\tau} + \gamma_{w,d,\tau} \leq 1 \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \tau \in \mathrm{T}''$$

Note that for a time slot $\bar{\tau}$ where $\theta_{w,d,\bar{\tau}} = 0$ because $\sum\limits_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t.\mathtt{Overlap}(\{\tau+1,\ldots,\tau+1+\mathtt{S}\})}} y_{c,t}$ is fixed to 0 (there exists no $y_{c,t}$ satisfying the sum conditions), then

$$\varphi_{w,d,\bar{\tau}} = \gamma_{w,d,\bar{\tau}} \qquad \forall\, w \in \mathcal{W}, d \in \mathcal{D}, \bar{\tau} \in \mathrm{T} : \left| \bigcap_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c: \\ t.\mathtt{Overlap}(\{\bar{\tau}+1,\ldots,\bar{\tau}+1+\mathtt{S}\})}} \{y_{c,t}\} \right| = 0$$

Special case
A special case is where a class length is longer than $\mathtt{M}$. We define the set of such classes as $\mathcal{C}_\delta^{>\mathtt{M}}$. For the hard constraint, we know that if a class $c \in \mathcal{C}_\delta^{>\mathtt{M}}$ is scheduled in a specific time, then any other class from $\mathcal{C}_\delta$ cannot use a time that will place the two in the same block. It implies that if the times share at least one week and at least one day, there cannot be less than $\mathtt{S}$ time slots between them.

$$\text{Hard:} \qquad y_{c_i,t_i} + \sum_{\substack{c_j \in \mathcal{C}_\delta \setminus \{c_i\}, \\ t_j \in \mathcal{T}_{c_j}: \\ t^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset \,\wedge \\ t^{\text{days}} \cap t_j^{\text{days}} \neq \emptyset \,\wedge \\ (t_i^{\text{start}} - t_j^{\text{end}} < \mathtt{S} \,\vee \\ t_j^{\text{start}} - t_i^{\text{end}} < \mathtt{S})}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i \in \mathcal{C}_\delta^{>\mathtt{M}}, t \in \mathcal{T}_{c_i}$$

For the soft constraint we need to set the $\rho_{w,d,\tau}$ variable if the time between the class $c_i \in \mathcal{C}_\delta^{>\mathtt{M}}$ and another class $c_j \in \mathcal{C}_\delta \setminus \{c_i\}$ is less than $\mathtt{S}$. That means that we must have a constraint for each week and day of any time of the class c_i.

$$\text{Soft:} \quad y_{c_i,t_i} + M \sum_{\substack{c_j \in \mathcal{C}_\delta \setminus \{c_i\}, \\ t_j \in \mathcal{T}_{c_j}: \\ w \in t_j^{\text{weeks}} \,\wedge \\ d \in t_j^{\text{days}} \,\wedge \\ (t_i^{\text{start}} - t_j^{\text{end}} < \mathtt{S} \,\vee \\ t_j^{\text{start}} - t_i^{\text{end}} < \mathtt{S})}} y_{c_j,t_j} \leq \rho_{w,d,t^{\text{start}}} \qquad \forall c_i \in \mathcal{C}_\delta^{>\mathtt{M}}, t \in \mathcal{T}_{c_i}, w \in t^{\text{weeks}}, d \in t^{\text{days}}$$

6.4.3 Student sectioning

Recall that $e_{s,c}$ is the decision variable that tells if student s is attending class c.

The number of students attending any class cannot exceed the limitation.

$$\sum_{s \in \mathcal{S}_c} e_{s,c} \leq c^{\text{limit}} \qquad \forall\, c \in \mathcal{C}$$

If a student attends a class given a parent class, the parent class must also be attended. Equality does not hold since classes can share the same parent.

$$e_{s,c_i} \leq e_{s,c_j} \qquad \forall\, s \in \mathcal{S}, c \in \mathcal{C}_s : c_i^{\text{parent}} = c_j$$

6.4.3.1 Students attending courses

For courses with only one configuration, the students who must attend the course must attend exactly one class from each configuration subparts.

$$\sum_{c \in \mathcal{C}_\zeta} e_{s,c} = 1 \qquad \forall\, k \in \mathcal{K}_s : |\Omega_k| = 1, \omega \in \Omega_k, \zeta \in \mathrm{Z}_\omega, s \in \mathcal{S}_k$$

For courses that have more than one configuration, we define the auxiliary variable $b_{s,\omega} \in \{0,1\}$.

$$b_{s,\omega} = \begin{cases} 1 & \text{if student } s \in \mathcal{S} \text{ is attending a class in configuration } \omega \in \Omega_k \\ 0 & \text{otherwise} \end{cases}$$

The students must attend the courses by attending exactly one of the configurations.

$$\sum_{\omega \in \Omega_k} b_{s,\omega} = 1 \qquad \forall\, s \in \mathcal{S}, k \in \mathcal{K}_s : |\Omega_k| > 1$$

To attend a configuration, the students must attend exactly one class from each of the configuration subparts. Furthermore, if a student is not attending a configuration, no classes from the subparts of that configuration can be attended.

$$\sum_{c \in \mathcal{C}_\zeta} e_{s,c} = b_{s,\omega} \qquad \forall\, k \in \mathcal{K}_s : |\Omega_k| > 1, \omega \in \Omega_k, \zeta \in \mathrm{Z}_\omega, s \in \mathcal{S}_k$$

6.4.3.2 Student conflicts

For student conflicts we define the variable $\chi_{s,c_i,c_j} \in \{0,1\}$

$$\chi_{s,c_i,c_j} = \begin{cases} 1 & \substack{\text{if there is a student conflict for student } s \in \mathcal{S} \\ \text{between classes } c_i \in \mathcal{C}_s \text{ and } c_j \in \mathcal{C}_s} \\ 0 & \text{otherwise} \end{cases}$$

This variable is dependent on variables f_{s,c_i,c_j} and o_{c_i,c_j}.

$$f_{s,c_i,c_j} = \begin{cases} 1 & \text{if student } s \in \mathcal{S} \text{ is attending both class } c_i \in \mathcal{C}_s \text{ and } c_j \in \mathcal{C}_s \\ 0 & \text{otherwise} \end{cases}$$

$$o_{c_i,c_j} = \begin{cases} 1 & \text{if classes } c_i \in \mathcal{C} \text{ and } c_j \in \mathcal{C} \text{ overlaps} \\ 0 & \text{otherwise} \end{cases}$$

Both have to be 1 for a student conflict to occur

$$o_{c_i,c_j} + f_{s,c_i,c_j} - 1 \leq \chi_{s,c_i,c_j} \qquad \forall\, s \in \mathcal{S}, (c_i, c_j) \in \mathcal{C}_s : i < j$$

Controlling the auxiliary variables

The variable f_{s,c_i,c_j} is dependent on the variables $e_{s,c}$

$$e_{s,c_i} + e_{s,c_j} - 1 \leq f_{s,c_i,c_j} \qquad \forall\, s \in \mathcal{S}, (c_i, c_j) \in \mathcal{C}_s : i < j$$

$$e_{s,c_i} \geq f_{s,c_i,c_j} \qquad \forall\, s \in \mathcal{S}, (c_i, c_j) \in \mathcal{C}_s : i < j$$

$$e_{s,c_j} \geq f_{s,c_i,c_j} \qquad \forall\, s \in \mathcal{S}, (c_i, c_j) \in \mathcal{C}_s : i < j$$

The variable o_{c_i,c_j} is dependent on the time and room of both classes. It is similar to the `SameAttendees` distribution constraint, but here it is split into two types of constraints, one for the times that overlap and one for the times that do not overlap, but the assigned rooms cause an overlap.

$$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i\texttt{.Overlap(}t_j\texttt{)}}} y_{c_j,t_j} - 1 \leq o_{c_i,c_j} \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in t_{c_i}$$

$$x_{c_i,t_i,r_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg t_i\texttt{.Overlap(}t_j\texttt{)}, \\ r_j \in \mathcal{R}_{c_j}: \\ t_i\texttt{.Overlap(}t_j\texttt{,}r_j\texttt{,}r_i\texttt{)}}} x_{c_j,t_j,r_j} - 1 \leq o_{c_i,c_j} \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in t_{c_i}, r_i \in \mathcal{R}_{c_i}$$

6.5 Objectives

There are four categories of objectives: time, room, distribution constraints, and student conflicts. These categories have respective weights ψ_t, ψ_r, ψ_δ, and ψ_s that prioritize the types of objectives compared to each other.

6.5.1 Time

The time objective is related to a penalty on a class c being scheduled at a specific time t.

$$\psi_t \sum_{\substack{c \in \mathcal{C}, \\ t \in \mathcal{T}_c}} p_{c,t} y_{c,t}$$

6.5.2 Room

The time objective is related to a penalty on a class c being assigned a specific room r.

$$\psi_r \sum_{\substack{c \in \mathcal{C}, \\ r \in \mathcal{R}_c}} p_{c,r} w_{c,r}$$

6.5.3 Distribution constraint

Recall that each soft distribution constraint defined a penalty variable p, which was written without dimensions and a penalty cost c_p. We have the set of all penalty variables $\mathcal{P}$.

$$\psi_\delta \sum_{p \in \mathcal{P}} c_p p$$

6.5.4 Student conflicts

Each student conflict has a penalty of 1; thus, we penalize the total number of student conflicts.

$$\psi_s \sum_{\substack{s \in \mathcal{S}, \\ (c_i, c_j) \in \mathcal{C}_s}} \chi_{s,c_i,c_j}$$

6.6 Results

The results consist of two parts. Section 6.6.1 presents the sizes of the MIP models in number of constraints and variables. Section 6.6.2 presents the performance of solving the MIP.
The MIP is solved using Gurobi 9.0 with 8 threads on a 64bit computer running Scientific Linux 7.7. The machine is equipped with two Intel Xeon E5-2650 v4 CPUs clocked at 2.20GHz and 256GB of RAM.
The 30 instances from the ITC 2019 competition are used to test the MIP. There is no data available for *pu-proj-fal19* as we could not construct the MIP model because of a lack of memory.

6.6.1 Size of the MIP

The total number of constraints and variables are shown in Table 6.4. Appendix 6.B gives specific details on the number of constraints and variables used to model the different parts of the MIP.

Instance	**Constraints**	**Variables**
agh-fis-spr17	10,716,234	5,270,957
agh-ggis-spr17	10,971,106	10,532,658
bet-fal17	7,427,922	4,770,806
iku-fal17	11,440,488	3,772,222
mary-spr17	1,716,685	1,287,460
muni-fi-spr16	4,958,093	4,350,531
muni-fsps-spr17	1,003,857	717,147
muni-pdf-spr16c	26,203,626	6,922,578
pu-llr-spr17	8,980,995	5,542,707
tg-fal17	597,723	232,749
agh-ggos-spr17	15,912,008	3,523,533
agh-h-spr17	12,642,901	1,881,442
lums-spr18	589,187	458,344
muni-fi-spr17	6,475,267	5,711,500
muni-fsps-spr17c	17,717,840	1,466,645
muni-pdf-spr16	10,554,793	6,901,019
nbi-spr18	1,955,308	631,823
pu-d5-spr17	31,570,758	30,624,014
pu-proj-fal19		
yach-fal17	4,407,666	1,336,902
agh-fal17	45,591,515	25,795,085
bet-spr18	10,006,170	6,165,343
iku-spr18	9,617,259	3,728,533
lums-fal17	525,567	448,222
mary-fal18	3,754,624	3,307,703
muni-fi-fal17	8,032,513	7,384,314
muni-fspsx-fal17	23,884,170	2,791,093
muni-pdfx-fal17	48,957,509	22,345,083
pu-d9-fal19	115,664,227	56,683,344
tg-spr18	645,009	130,533

Table 6.4: Total number of constraints and variables for each instance.

6.6.2 Solving the MIP

The performance is tested with a 24 hour time limit, including reading and processing the data. Table 6.5 shows the performance of the MIP after 1 hour and 24 hours.

Time	**1 hour**			**24 hours**		
Instance	UB	LB	Gap	UB	LB	Gap
agh-fis-spr17	-	-	-	-	779	100%
agh-ggis-spr17	-	-	-	-	21,703	100%
bet-fal17	-	-	-	-	-	-
iku-fal17	-	-	-	-	10,947	100%
mary-spr17	-	2,435	100%	15,932	13,212	17.07%
muni-fi-spr16	-	-	-	-	3,276	100%
muni-fsps-spr17	-	851	100%	**868**	868	0%
muni-pdf-spr16c	-	-	-	-	9,196	100%
pu-llr-spr17	-	-	-	10,710	9,683	9.59%
tg-fal17	**4,215**	4,215	0%	**4,215**	4,215	0%
agh-ggos-spr17	-	-	-	-	-	-
agh-h-spr17	-	-	-	-	5	100%
lums-spr18	-	1	100%	95	24	74.74%
muni-fi-spr17	-	-	-	24,572	2,056	91.63%
muni-fsps-spr17c	-	-	-	-	923	100%
muni-pdf-spr16	-	-	-	-	-	-
nbi-spr18	18,979	17,438	8.12%	18,212	17,654	3.06%
pu-d5-spr17	-	-	-	-	4,147	100%
pu-proj-fal19						
yach-fal17	-	30	100%	19,046	516	97.29%
agh-fal17	-	-	-	-	-	-
bet-spr18	-	-	-	-	-	-
iku-spr18	-	-	-	-	14,006	100%
lums-fal17	-	196	100%	405	233	42.47%
mary-fal18	-	-	-	-	3,009	100%
muni-fi-fal17	-	-	-	-	1,486	100%
muni-fspsx-fal17	-	-	-	-	1,680	100%
muni-pdfx-fal17	-	-	-	-	-	-
pu-d9-fal19	-	-	-	-	-	-
tg-spr18	**12,704**	12,704	0%	**12,704**	12,704	0%

Table 6.5: Runtime stats after 1 hour and 24 hours. Optimal solutions are in bold. Dash means that the solver is running, but there was no value.

When solving the MIP, Gurobi will first try to reduce the MIP by a presolve procedure. The presolve procedure will try to make the MIP smaller and easier to solve. In Table 6.6, we present the number of constraints and variables removed by presolve and the size of the model after presolve. The Gurobi Presolve parameter is left at its default setting.

Instance	Removed constraints	Removed variables	Number of constraints after presolve	Number of variables after presolve	Time (sec)
agh-fis-spr17	3,303,883	2,688,938	7,412,351	2,582,019	2,956
agh-ggis-spr17	9,004,781	8,571,540	1,966,325	1,961,118	239
bet-fal17	1,208,055	788,486	6,219,867	3,982,320	680
iku-fal17	2,040,648	478,300	9,399,840	3,293,922	2,662
mary-spr17	942,132	761,533	774,553	525,927	62
muni-fi-spr16	1,843,183	1,644,797	3,114,910	2,705,734	100
muni-fsps-spr17	471,269	349,566	532,588	367,581	34
muni-pdf-spr16c	9,263,603	3,277,787	16,940,023	3,644,791	4,825
pu-llr-spr17	4,288,999	3,652,660	4,691,996	1,890,047	289
tg-fal17	477,409	106,435	120,314	126,314	22
agh-ggos-spr17	2,557,583	873,066	13,354,425	2,650,467	1,413
agh-h-spr17	1,830,684	295,957	10,812,217	1,585,485	5,379
lums-spr18	101,420	22,924	487,767	435,420	189
muni-fi-spr17	2,050,479	1,833,495	4,424,788	3,878,005	150
muni-fsps-spr17c	4,574,019	604,346	13,143,821	862,299	1,497
muni-pdf-spr16	2,739,129	2,298,089	7,815,664	4,602,930	698
nbi-spr18	734,527	431,815	1,220,781	200,008	52
pu-d5-spr17	23,427,922	22,939,347	8,142,836	7,684,667	473
pu-proj-fal19					
yach-fal17	1,086,201	326,048	3,321,465	1,010,854	194
agh-fal17	14,037,069	12,331,961	31,554,446	13,463,124	11,506
bet-spr18	1,662,060	971,980	8,344,110	5,193,363	660
iku-spr18	2,666,485	780,152	6,950,774	2,948,381	2,216
lums-fal17	109,252	16,101	416,315	432,121	134
mary-fal18	1,438,431	1,294,436	2,316,193	2,013,267	92
muni-fi-fal17	2,625,159	2,433,535	5,407,354	4,950,779	154
muni-fspsx-fal17	5,813,525	1,197,584	18,070,645	1,593,509	2,860
muni-pdfx-fal17	22,427,991	14,035,002	26,529,518	8,310,081	11,008
pu-d9-fal19	39,699,909	32,863,017	75,964,318	23,820,327	5,970
tg-spr18	573,696	58,854	71,313	71,679	29

Table 6.6: The number constraints and variable removed by Gurobi presolve and the MIP model sizes after presolve. Last column is the time spend in presolve.

6.7 Concluding remarks

We have presented a MIP formulation for the ITC 2019 problem. This MIP formulation results in models with a vast number of constraints and variables. The performance was tested on the 30 instances of ITC 2019 resulting in solutions to ten instances, including three optimal solutions. The results also provide lower bounds on 22 of the instances. One instance reached the memory limit and thus was not able to be computed. The results are, of course, dependant on different Gurobi parameter settings. For example, one could wish to use a more aggressive presolve strategy and a MIP focus parameter set to *feasibility* or *optimality*. Such settings might provide more reductions in presolve and might also provide feasible solutions to more instances with worse lower bounds as a consequence. However, before such parameter tuning is performed, it should be considered if the model can be formulated differently. Table 6.6 shows the number of constraints and variables removed by presolve. For some instances, Gurobi can remove numerous constraints and variables in very little time. This might indicate that the MIP formulation contains many redundant constraints and/or variables.

This basic MIP formulation can provide good solutions to smaller instances within a reasonable time. There is no time limit of the ITC 2019, but the final data instances are released 10 days before the deadline. It is not expected that the MIP models will perform significantly better by increasing the time limit to 10 days. We believe that it is likely that an improved formulation of the MIP or a matheuristic based on the MIP can outperform these results.

Appendices

6.A The `Overlap` function

The `Overlap` function takes different input parameters and gives a boolean result if the time and the input parameters overlap.

The different types

t`.Overlap(`$\{\tau_{\text{min}}, \ldots, \tau_{\text{max}}\}$`)`
- Returns *true* if the time slots of t overlaps the time slots from τ_{min} to τ_{max}.

$$\tau_{\text{min}} < t^{\text{end}} \quad \wedge \quad t^{\text{start}} < \tau_{\text{max}}$$

t`.Overlap(`$\bar{t}$`)`
- Returns *true* if the times t and $\bar{t}$ have at least one week and one day in common and the time slots overlaps.

$$\begin{aligned} t^{\text{weeks}} \cap \bar{t}^{\text{weeks}} \neq \emptyset \quad & \wedge \\ t^{\text{days}} \cap \bar{t}^{\text{days}} \neq \emptyset \quad & \wedge \\ \bar{t}^{\text{start}} < t^{\text{end}} \quad \wedge \quad t^{\text{start}} < \bar{t}^{\text{end}} & \end{aligned}$$

t`.Overlap(`$\bar{t}, \bar{r}, r$`)` - Returns *true* if the times t and $\bar{t}$ have at least one week and one day in common and the time slots overlaps or it is not possible to get from one room to the other in the time difference.

$$\begin{aligned} t^{\text{weeks}} \cap \bar{t}^{\text{weeks}} \neq \emptyset \quad & \wedge \\ t^{\text{days}} \cap \bar{t}^{\text{days}} \neq \emptyset \quad & \wedge \\ [\bar{t}^{\text{start}} < t^{\text{end}} \quad \wedge \quad t^{\text{start}} < \bar{t}^{\text{end}} \quad & \vee \\ \max\{t^{\text{start}}, \bar{t}^{\text{start}}\} - \min\{t^{\text{end}}, \bar{t}^{\text{end}}\} < \texttt{distance}(\bar{r}, r)] & \end{aligned}$$

6.B Specified MIP size

Tables 6.7-6.11 gives the specified number of constraints used to model each of the distribution constraint types. Table 6.12 gives the number of constraint and variable used for student sectioning.

Instance	**SameStart**	**SameTime**	**SameDays**	**SameWeeks**	**SameRoom**
agh-fis-spr17	0 | 0	6,002 | 341	6,002 | 9,657	731 | 0	68 | 51
agh-ggis-spr17	0 | 0	8,281 | 397	15,871 | 21,048	0 | 0	1,094 | 126
bet-fal17	64 | 0	37 | 0	198 | 55,680	0 | 0	333 | 510
iku-fal17	208 | 0	3,324 | 0	13,628 | 1,714	0 | 0	4,959 | 1,677
mary-spr17	0 | 0	164 | 0	659 | 875	0 | 0	159 | 57
muni-fi-spr16	0 | 0	628 | 0	1,253 | 278	84 | 0	149 | 9
muni-fsps-spr17	0 | 0	2,308 | 1,192	2,558 | 1,222	0 | 0	116 | 15
muni-pdf-spr16c	0 | 0	3,626 | 573	3,904 | 0	0 | 0	132 | 28,555
pu-llr-spr17	1,736 | 0	261 | 0	296 | 563	0 | 0	427 | 277
tg-fal17	0 | 0	833 | 0	833 | 6,967	0 | 0	0 | 0
agh-ggos-spr17	0 | 0	19,938 | 0	21,962 | 516	44 | 0	789 | 1
agh-h-spr17	0 | 0	32,138 | 616	32,218 | 102,452	54 | 0	271 | 27
lums-spr18	0 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-spr17	0 | 0	300 | 0	1,057 | 65	162 | 0	141 | 11
muni-fsps-spr17c	0 | 0	1,056 | 0	1,182 | 0	846 | 234	0 | 1,489
muni-pdf-spr16	0 | 0	1,723 | 40	2,289 | 5,891	0 | 0	619 | 2,387
muni-pdf-spr16c	0 | 0	947 | 0	992 | 275	0 | 0	0 | 0
pu-d5-spr17	4 | 0	75 | 178	2,016 | 323	0 | 0	671 | 1,965
pu-proj-fal19					
yach-fal17	0 | 0	0 | 0	23 | 0	0 | 0	24 | 1,145
agh-fal17	0 | 0	44,698 | 65,737	51,903 | 153,694	1,592 | 12	3,434 | 2,782
bet-spr18	2 | 0	316 | 0	831 | 69,564	0 | 0	249 | 1,417
iku-spr18	210 | 0	3,972 | 0	14,822 | 50	0 | 0	5,518 | 0
lums-fal17	0 | 0	77 | 0	118 | 0	0 | 0	0 | 0
mary-fal18	0 | 0	115 | 0	408 | 206	0 | 0	354 | 46
muni-fi-fal17	0 | 0	388 | 0	788 | 448	0 | 0	289 | 0
muni-fspsx-fal17	0 | 0	615 | 0	1,987 | 1,372	1,320 | 0	69 | 2,235
muni-pdfx-fal17	0 | 0	7,571 | 655	12,135 | 7,439	392 | 444	898 | 42,685
pu-d9-fal19	1,512 | 0	1,022 | 0	5,413 | 4,965	0 | 0	5,149 | 8,068
tg-spr18	0 | 0	180 | 17,591	350 | 17,591	0 | 0	1,081 | 3,871

Table 6.7: Number of constraints in the MIP to model each of the distribution constraint types. Data represented Hard | Soft.

Instance	DifferentTime	DifferentDays	DifferentWeeks	DifferentRoom
agh-fis-spr17	0 | 0	478 | 210	0 | 0	0 | 0
agh-ggis-spr17	0 | 0	78 | 14	0 | 0	0 | 0
bet-fal17	3 | 0	8,154 | 54,691	0 | 0	0 | 431
iku-fal17	0 | 0	105 | 0	0 | 0	0 | 0
mary-spr17	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-spr16	0 | 0	171 | 225	0 | 0	0 | 0
muni-fsps-spr17	0 | 0	50 | 0	0 | 0	0 | 0
muni-pdf-spr16c	0 | 0	0 | 0	178,622 | 1,512	0 | 0
pu-llr-spr17	38 | 19	77 | 51	0 | 0	0 | 0
tg-fal17	0 | 0	0 | 331	0 | 0	0 | 0
agh-ggos-spr17	0 | 0	25 | 0	0 | 0	0 | 0
agh-h-spr17	0 | 0	0 | 0	0 | 0	0 | 0
lums-spr18	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-spr17	0 | 0	146 | 162	0 | 0	8 | 0
muni-fsps-spr17c	0 | 0	0 | 0	118,192 | 0	0 | 0
muni-pdf-spr16	0 | 0	664 | 1,176	116 | 0	0 | 0
muni-pdf-spr16c	0 | 0	1,333 | 40	0 | 0	0 | 0
pu-d5-spr17	0 | 0	2,303 | 0	0 | 0	0 | 0
pu-proj-fal19				
yach-fal17	0 | 0	12,427 | 0	0 | 0	0 | 0
agh-fal17	0 | 0	1,101 | 0	0 | 0	0 | 0
bet-spr18	0 | 0	11,507 | 68,076	0 | 0	0 | 1,924
iku-spr18	0 | 0	1,117 | 0	0 | 0	0 | 0
lums-fal17	0 | 0	0 | 0	0 | 0	0 | 0
mary-fal18	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-fal17	0 | 0	106 | 272	0 | 0	0 | 0
muni-fspsx-fal17	0 | 0	8 | 60	156,245 | 0	0 | 0
muni-pdfx-fal17	0 | 0	582 | 357	225,539 | 0	0 | 0
pu-d9-fal19	467 | 28	47 | 68	0 | 0	0 | 0
tg-spr18	0 | 0	1,359 | 0	0 | 0	0 | 0

Table 6.8: Number of constraints in the MIP to model each of the distribution constraint types. Data represented Hard | Soft.

Instance	**Overlap**	**NotOverlap**	**SameAttendees**	**Precedence**
agh-fis-spr17	0 | 0	2,286 | 222	6,263,527 | 3,374	272 | 22,983
agh-ggis-spr17	0 | 0	64 | 5,427	498,878 | 0	991 | 8,125
bet-fal17	0 | 0	205 | 0	1,521,402 | 0	0 | 0
iku-fal17	1,072 | 0	539,015 | 1,043	10,604,393 | 2,920	1,571 | 21,855
mary-spr17	0 | 0	0 | 16,385	306,332 | 539,511	41 | 0
muni-fi-spr16	0 | 0	2,700 | 17,279	57,483 | 0	464 | 378
muni-fsps-spr17	0 | 0	0 | 6,215	94,501 | 0	957 | 0
muni-pdf-spr16c	0 | 0	0 | 29,354	18,099,414 | 0	91,893 | 1,265
pu-llr-spr17	0 | 0	3,111 | 5,643	37,641 | 0	0 | 126
tg-fal17	0 | 0	7,406 | 0	407,532 | 134,745	0 | 0
agh-ggos-spr17	0 | 0	0 | 0	3,161,309 | 0	989 | 67,366
agh-h-spr17	0 | 0	159,239 | 0	8,445,889 | 12	2,352 | 1,104
lums-spr18	0 | 0	108,029 | 33,772	392,946 | 0	0 | 0
muni-fi-spr17	0 | 0	3,539 | 16,670	73,599 | 0	1,208 | 84
muni-fsps-spr17c	0 | 0	0 | 9,994	2,979,413 | 0	16,321 | 0
muni-pdf-spr16	0 | 0	0 | 73,914	2,321,086 | 0	454 | 201
muni-pdf-spr16c	0 | 0	0 | 635	334,989 | 0	0 | 0
pu-d5-spr17	0 | 0	7,261 | 90,598	55,526 | 2	965 | 1,519
pu-proj-fal19				
yach-fal17	0 | 0	0 | 12,075	339,540 | 0	0 | 0
agh-fal17	0 | 0	10,151 | 1,402	10,410,795 | 234,335	6,080 | 190,573
bet-spr18	0 | 0	180 | 960	2,254,545 | 0	0 | 0
iku-spr18	0 | 0	766,819 | 0	8,521,716 | 45,806	1,799 | 17,797
lums-fal17	0 | 0	153,234 | 31,524	287,927 | 0	0 | 0
mary-fal18	0 | 0	0 | 13,907	111,775 | 11,781	198 | 0
muni-fi-fal17	0 | 0	1,901 | 12,557	53,199 | 0	421 | 1,129
muni-fspsx-fal17	0 | 0	29,711 | 26,530	3,425,746 | 11	26,794 | 532
muni-pdfx-fal17	0 | 0	372 | 72,744	25,140,403 | 2,653	115,569 | 6,390
pu-d9-fal19	0 | 0	3,937 | 78,834	418,931 | 0	76 | 759
tg-spr18	0 | 0	3,860 | 0	559,997 | 0	120 | 356

Table 6.9: Number of constraints in the MIP to model each of the distribution constraint types. Data represented Hard | Soft.

Instance	**WorkDay**	**MinGap**	**MaxDays**
agh-fis-spr17	0 | 1,844	0 | 0	80 | 48
agh-ggis-spr17	6,449 | 2,104	0 | 0	0 | 24
bet-fal17	9,757 | 168	0 | 55,351	80 | 0
iku-fal17	10,027 | 642	0 | 0	0 | 0
mary-spr17	201 | 112	108 | 739	0 | 0
muni-fi-spr16	713 | 162	0 | 0	0 | 0
muni-fsps-spr17	130 | 8	0 | 0	0 | 0
muni-pdf-spr16c	28 | 61,001	0 | 0	0 | 0
pu-llr-spr17	17 | 459	0 | 44	0 | 0
tg-fal17	0 | 118	0 | 0	0 | 0
agh-ggos-spr17	1,821 | 75	0 | 0	16 | 16
agh-h-spr17	74 | 2,371	0 | 0	136 | 40
lums-spr18	0 | 0	0 | 0	0 | 0
muni-fi-spr17	638 | 58	0 | 0	8 | 0
muni-fsps-spr17c	100 | 0	0 | 0	0 | 0
muni-pdf-spr16	488 | 13,459	33 | 210	0 | 0
muni-pdf-spr16c	0 | 0	0 | 275	0 | 0
pu-d5-spr17	1,366 | 143	85 | 54	0 | 0
pu-proj-fal19			
yach-fal17	0 | 0	23 | 0	0 | 0
agh-fal17	5,127 | 6,424	16 | 104	352 | 216
bet-spr18	13,363 | 252	0 | 68,769	120 | 0
iku-spr18	11,174 | 0	0 | 0	0 | 0
lums-fal17	0 | 0	0 | 0	0 | 0
mary-fal18	202 | 45	0 | 116	0 | 0
muni-fi-fal17	391 | 263	0 | 0	16 | 0
muni-fspsx-fal17	592 | 52	0 | 1,364	0 | 0
muni-pdfx-fal17	1,069 | 21,761	0 | 1,132	0 | 64
pu-d9-fal19	3,370 | 1,430	199 | 95	0 | 0
tg-spr18	0 | 0	0 | 0	0 | 0

Table 6.10: Number of constraints in the MIP to model each of the distribution constraint types. Data represented Hard | Soft.

Instance	**MaxDayLoad**	**MaxBreaks**	**MaxBlock**
agh-fis-spr17	$0 \mid 0$	$0 \mid 93,490$	$0 \mid 0$
agh-ggis-spr17	$0 \mid 0$	$0 \mid 5,904$	$0 \mid 0$
bet-fal17	$32 \mid 0$	$0 \mid 0$	$749,119 \mid 0$
iku-fal17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
mary-spr17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-fi-spr16	$114 \mid 0$	$0 \mid 0$	$0 \mid 14,325$
muni-fsps-spr17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-pdf-spr16c	$50 \mid 1,289$	$0 \mid 0$	$6,992 \mid 211,584$
pu-llr-spr17	$0 \mid 0$	$0 \mid 6,032$	$0 \mid 0$
tg-fal17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
agh-ggos-spr17	$0 \mid 0$	$9,898 \mid 0$	$0 \mid 0$
agh-h-spr17	$546 \mid 71$	$0 \mid 431,581$	$0 \mid 0$
lums-spr18	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-fi-spr17	$70 \mid 0$	$0 \mid 0$	$91,638 \mid 24,893$
muni-fsps-spr17c	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-pdf-spr16	$286 \mid 1,541$	$0 \mid 0$	$26,352 \mid 554,275$
muni-pdf-spr16c	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
pu-d5-spr17	$0 \mid 0$	$41,250 \mid 0$	$0 \mid 0$
pu-proj-fal19			
yach-fal17	$2,935 \mid 0$	$0 \mid 0$	$0 \mid 0$
agh-fal17	$975 \mid 0$	$165 \mid 792,455$	$0 \mid 0$
bet-spr18	$0 \mid 0$	$0 \mid 0$	$892,608 \mid 0$
iku-spr18	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
lums-fal17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
mary-fal18	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-fi-fal17	$104 \mid 53$	$0 \mid 0$	$148,171 \mid 27,939$
muni-fspsx-fal17	$0 \mid 0$	$0 \mid 0$	$0 \mid 0$
muni-pdfx-fal17	$428 \mid 1,614$	$0 \mid 0$	$6,145 \mid 562,289$
pu-d9-fal19	$0 \mid 31$	$56,607 \mid 3,016$	$0 \mid 0$
tg-spr18	$0 \mid 152$	$0 \mid 0$	$0 \mid 0$

Table 6.11: Number of constraints in the MIP to model each of the distribution constraint types. Data represented Hard | Soft.

The number of constraints and variables used to model the student sectioning is presented in Table 6.12. The constraints and variables are split into two categories. One category is the student sectioning part. This part considers the constraints and variables used in assigning the students to the correct classes. The other part, student conflicts, is the constraints and variables used to model when the students' assignment leads to conflict.

	Student Sectioning			**Student Conflicts**	
Instance	$e_{s,c}$	$b_{s,\omega}$	**Constraints**	**Variables**	**Constraints**
agh-fis-spr17	65,376	0	53,277	3,566,172	3,986,367
agh-ggis-spr17	116,552	0	57,723	10,061,361	10,257,182
bet-fal17	94,893	15,990	62,440	3,575,690	4,843,252
iku-fal17	-	-	-	-	-
mary-spr17	28,230	0	6,046	507,735	806,431
muni-fi-spr16	64,506	14	7,454	4,205,866	4,834,674
muni-fsps-spr17	20,667	776	7,704	644,280	867,989
muni-pdf-spr16c	80,918	5,246	56,494	3,735,530	7,157,505
pu-llr-spr17	236,980	20,353	103,598	5,108,450	8,773,805
tg-fal17	-	-	-	-	-
agh-ggos-spr17	56,818	0	45,621	2,111,511	12,415,681
agh-h-spr17	11,530	0	4,954	310,548	2,996,086
lums-spr18	-	-	-	-	-
muni-fi-spr17	70,257	60	8,373	5,504,735	6,232,852
muni-fsps-spr17c	17,190	0	10,135	973,959	14,490,356
muni-pdf-spr16	103,864	2,822	22,982	5,468,914	7,420,773
muni-pdf-spr16c	29,986	0	4,984	422,248	1,565,002
pu-d5-spr17	442,232	9,886	302,079	29,956,847	31,033,720
pu-proj-fal19					
yach-fal17	28,779	150	25,665	1,183,118	3,986,219
agh-fal17	305,022	108	198,075	20,701,326	32,709,410
bet-spr18	105,021	14,998	80,461	4,814,300	6,473,132
iku-spr18	-	-	-	-	-
lums-fal17	-	-	-	-	-
mary-fal18	83,731	0	12,970	2,986,780	3,564,777
muni-fi-fal17	90,997	0	9,991	7,141,048	7,758,047
muni-fspsx-fal17	44,916	194	18,373	2,155,918	20,066,497
muni-pdfx-fal17	241,256	14,776	135,103	16,817,990	22,120,722
pu-d9-fal19	941,391	57,306	513,531	54,867,830	114,406,242
tg-spr18	-	-	-	-	-

Table 6.12: Overview of the number of variables and constraints that are connected to student sectioning and student conflicts.

References

Müller, T., Rudová, H., and Müllerová, Z. (2018). "University course timetabling and international timetabling competition 2019". In: *Proceedings of* *the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria*. Ed. by E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, pp. 5–31.

Chapter 7

A Graph-Based MIP Formulation of the International Timetabling Competition 2019

Dennis Søren Holm[a] · Rasmus Ørnstrup Mikkelsen[a,b] · Matias Sørensen[b] · Thomas Jacob Riis Stidsen[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]MaCom A/S, Vesterbrogade 48, 1., DK-1620 København V, Denmark

Status: Submitted to Journal of Scheduling

Abstract: The International Timetabling Competition 2019 posed a university timetabling problem involving assigning classes to times and rooms for an entire semester while assigning students to required classes. To solve this problem, we propose a new mixed integer programming (MIP) formulation of the problem. The MIP formulation takes advantage of different graph structures in the conflict graphs to construct a strong formulation of the constraints. In addition, we introduce a reduction algorithm that removes redundancies from the input data. We show that the reduction algorithm, combined with the new MIP formulation, outperforms the MIP formulated by Holm et al. (2020) and thus, becomes the new state-of-the-art MIP formulation for the ITC 2019. This paper reports the MIP formulation that we used during the 2019 competition and discusses additional approaches that one can use to strengthen the MIP.

Keywords: Mixed Integer Programming · Conflict Graph · Clique Cover · University Timetabling · International Timetabling Competition 2019 · ITC 2019

7.1 Introduction

The International Timetabling Competition 2019 (ITC 2019) is the fourth such competition. As these competitions encourage research in both the theory and practice of automated timetabling, they are essential to the field of timetabling. The first International Timetabling Competition (ITC2002) focused on a simplification of a typical university course timetabling problem (UCTTP) (Paechter et al., 2002). The second one, ITC2007, had three different university timetabling problems: examination timetabling, post enrollment-based course timetabling, and curriculum-based course timetabling (CCT) (McCollum et al., 2010). The third competition, ITC2011, focused on high school timetabling (Post et al., 2016). In contrast, the novelty of ITC 2019 is that it focuses on the combination of UCTTP and student sectioning. In addition, it (a) presents a generalized model formulation that takes into account a diverse set of timetabling characteristics and (b) provides real-world data from universities worldwide (Müller et al., 2018).

The UCTTP is a $\mathcal{NP}$-hard combinatorial problem. Holm et al. (2020) show a basic mixed integer programming (MIP) formulation that can provide solutions to 10 out of the 30 ITC 2019 instances within 24 hours, including three optimal solutions. They also show that the MIP models have a huge number of constraints and variables. To reduce the number of these constraints and variables, and to strengthen the MIP formulation and improve the solvability, this paper proposes a new graph-based MIP model.

Several studies have shown that performing reductions on integer programming (IP) models can improve the models' solvability. Burke et al. (2008), Burke et al. (2010b), Burke et al. (2010a), and Burke et al. (2012) describe an IP model for the ITC2007 CCT problem called the monolithic formulation, which includes binary decision variables with indices, rooms, periods, and courses. They find that solving the monolithic model is difficult and provide various methods for overcoming the difficulty. Burke et al. (2010a) improve the IP formulation by contracting the lectures for each course into supernodes, thereby decreasing the number of variables and constraints. The supernode transformation is a symmetry-breaking method. Using the fact that because all the events in a course are equal, what matters is only that any one of those events is assigned the room and period, not any specific event in particular. Burke et al. (2012) develop a branch-and-cut algorithm based on the monolithic formulation. The branch-and-cut algorithm leaves out some objective costs, instead adding them dynamically as cuts. Furthermore, they add clique cuts and present some valid inequalities.

Bagger et al. (2019b), by replacing the curriculum conflict and teacher conflict constraints with clique constraints, strengthen the three-index MIP formulation of the ITC2007 CCT problem. They provide two formulations—one minimum cost flow-based the other multi-commodity flow-based—and are therefore able to reduce the number of variables and constraints of the models. Bagger et al. (2019a) use a pattern formulation of the ITC2007 CCT problem, in which each pattern represents an assignment of events to time slots, independent of the days of the week. Their preprocessing ensures that invalid patterns for some courses are not generated as variables in the model. They also introduce and use pattern conflict graphs to generate valid inequalities in the form of clique constraints. Both Burke et al. (2008), Burke et al. (2010b), Burke et al. (2010a), and Burke et al. (2012) and Bagger et al. (2019a) and Bagger et al. (2019b) show that by inspecting their timetable problem with a graph view, they can improve the formulations and solution approaches.

The usage of conflict graphs and clique constraints is beneficial in more areas than educational timetabling. Böðvarsdottir et al. (2019) use conflict graphs and cliques to model the Danish Nurse Rostering problem. Each vertex represents a triplet of an employee, a day, and a shift type in the conflict graph. They generate the conflict graph by pairwise checking vertices for whether a conflict exists or not. Thereby, they reduced the number of constraints by using a clique cover to model the constraints as clique constraints.

The MIP formulation in this present paper consists of a reduction of the data instances and a strong formulation of the MIP constraints. Given that the data contains redundant information, the reduction of the data instances entails reducing the raw input data. While the MIP reformulation primarily involves considering graph approaches to strengthen the formulation, it also takes advantage of implicitly understood structure. The paper is organized as follows: Section 7.2 briefly describes the problem. Section 7.3 presents the decision variables. Section 7.4 provides background theory on conflict graphs and different types of edge covers. Section 7.5 explains the reduction of the data instances. Sections 7.6, 7.7, and 7.8 explain the formulation of the MIP, the distribution constraints, the student section, and the objective function, respectively. Section 7.9 gives the results. Section 7.10 discusses future work and Section 7.11 concludes.

7.2 Problem description

The ITC 2019 problem formulation can be split into two connected parts. The scheduling part, where classes should be assigned a time and a room according to some constraints. The classes should also be scheduled such that they do not conflict in the matter of the defined distribution constraints. The distribution constraints restrict how the classes can be scheduled according to each other. The problem formulation contains 19 different distribution constraint types that describe features from assigning classes to equal times to restricting the total daily workload. The other part of the problem is student sectioning, where students must be assigned classes according to a set of courses that they must attend. The assignment of a student to a course should follow a defined course structure. For a comprehensive description of the problem, we refer to Müller et al. (2018).

7.3 Decision variables

The model uses two main binary decision variables; the scheduling variable $x_{c,t,r}$ and the student sectioning variable $e_{s,c}$. We define the scheduling variable as binary with indices class, time, and room.

$$x_{c,t,r} = \begin{cases} 1 & \text{if class } c \text{ is scheduled at time } t \text{ in room } r \\ 0 & \text{otherwise} \end{cases}$$

$x_{c,t,r}$ is defined for all $c \in \mathcal{C}$, $t \in \mathcal{T}_c$ and $r \in \mathcal{R}_c$.
The scheduling variables $x_{c,t,r}$ (class-time-room) leads to the following binary auxiliary variables $y_{c,t}$ (class-time) and $w_{c,r}$ (class-room).

$$y_{c,t} = \begin{cases} 1 & \text{if class } c \text{ is scheduled at time } t \\ 0 & \text{otherwise} \end{cases}$$

$$w_{c,r} = \begin{cases} 1 & \text{if class } c \text{ is scheduled in room } r \\ 0 & \text{otherwise} \end{cases}$$

We define the student sectioning variable $e_{s,c}$ as

$$e_{s,c} = \begin{cases} 1 & \text{if student } s \text{ is attending class } c \\ 0 & \text{otherwise} \end{cases}$$

$e_{s,c}$ is defined for all $s \in \mathcal{S}$ and $c \in \mathcal{C}_s$.

7.4 Conflict graphs

A conflict graph $G(\mathcal{V}, \mathcal{E})$ is a simple graph where each vertex represents a binary variable in the MIP model, i.e. $x_{c,t,r}$. We distinguish between a *hard constraint conflict graph* and a *soft constraint conflict graph*. There is an edge between two vertices in the hard constraint conflict graph if the corresponding variables are conflicting and thus not allowed simultaneously in a feasible solution. There is an edge between two vertices in the soft constraint conflict graph if choosing both corresponding variables is associated with a penalty. We model the penalty as the weight on the corresponding edge.

Using conflict graphs can simplify the process of mathematical modeling of the ITC 2019 XML format. Since the graphs are simple, at most one edge can be added between two vertices, and modeling each edge as a constraint is sufficient to cover all the constraints used to generate the conflict graph. However, this approach can be improved by taking advantage of some graph structures and finding an edge cover[1] of the conflict graph. We will present such structures later.

[1]Hereafter any *cover* is an edge cover since a vertex cover does not make sense in this setting.

The conflict graphs can be created of different types. A class-time conflict graph considering the $y_{c,t}$ variables, a class-room conflict graph considering the $w_{c,r}$ variables and a class-time-room conflict graph considering the $x_{c,t,r}$ variables. The latter is of course the largest of the three.
We introduce the different conflict graphs used in Section 7.4.1-7.4.3 and the different edge covers in Section 7.4.4-7.4.8.

7.4.1 Class-time conflict graphs

The class-time conflict graph exists in both a hard constraint and a soft constraint version. Both versions contains edges from many different distribution constraints, see Table 7.1.

Constraint	Opposite
`SameStart`	
`SameTime`	`DifferentTime`
`SameDays`	`DifferentDays`
`SameWeeks`	`DifferentWeeks`
`Overlap`	`NotOverlap`
`Precedence`	
`Workday(S)`	
`MinGap(G)`	

Table 7.1: Distribution constraints providing edges to the class-time conflict graph.

The constraints of Table 7.1 are the distribution constraints that consider only the time assignment of a class and can be validated in pairs, as shown in Müller et al. (2018). It is simple to check if there is an edge between two vertices (a pair of $y_{c,t}$), and if the constraint is soft, it is also easy to see what the edge weight is by a summation of the violated constraints' penalties.

7.4.1.1 Hard constraint class-time conflict graph

The hard constraint class-time conflict graph contains additional edges from more distribution constraints, see Table 7.2 for an overview. As it might not be clear which edges are added, a description of each distribution constraint follows.

`SameRoom`: For a pair of classes that must use the same room, an edge is added between the vertices if the times overlap, since assigning the classes overlapping times would result in a room double booking constraint violation.

`SameAttendees`: We divide the constraint into two parts. One part considering only the overlapping times ($y_{c,t}$), and another part considering only times that do not overlap but can cause a violation by the distance of the assigned rooms ($x_{c,t,r}$). The hard constraint class-time conflict graph contains edges on class-time pair that have overlapping times.

`MaxDays(D)`: For a pair of classes that can use no more than `D` days, an edge is added between the vertices if the number of different days of the times exceeds `D`.

`MaxDayLoad(S)`: For a class-time pair that has at least one week and one day in common, add an edge if the sum of the lengths is greater than `S`.

`MaxBlock(M, S)`: For pair of classes that cannot be scheduled together such that the block is longer than `M`, an edge is added between the vertices if the times are considered to be in the same block (duration between the times is less than `S`), and that block's length is greater than `M`.

Furthermore, we add edges between two vertices of the same class.

Distribution constraint	Condition
`SameRoom`	t_1 overlaps t_2
`SameAttendees`	t_1 overlaps t_2
`MaxDays(D)`	$\left\|t_1^{\text{days}} \cup t_2^{\text{days}}\right\| > D$
`MaxDayLoad(S)`	t_1 overlaps on days and weeks t_2 and $t_1^{\text{length}} + t_2^{\text{length}} > S$
`MaxBlock(M,S)`	t_1 and t_2 are in a block, which has a length greater than M

Table 7.2: Distribution constraints contributing with additional edges to the hard constraint class-time conflict graph. The condition considers two vertices $v_1 = \{v_{c_1}, v_{t_1}\}$ and $v_2 = \{v_{c_2}, v_{t_2}\}$; if the condition is met, there is an edge between the two vertices.

7.4.1.2 Soft constraint class-time conflict graph

The soft constraint class-time conflict graph contains the edges described in Section 7.4.1 and edges from the soft `SameAttendees` distribution constraints as described for the hard constraint class-time conflict graph. If an edge exists in the hard constraint class-time conflict graph, a similar edge will not be added in the soft constraint class-time conflict graph as it is redundant.

7.4.2 Class-room conflict graphs

The class-room conflict graph exists only as a hard constraint version. It considers less distribution constraints compared to the class-time conflict graph, see Table 7.3.
The distribution constraints of Table 7.3 are the constraints that consider the room assignment of a class and can be validated in pairs, as showed in Müller

Constraint	Opposite
`SameRoom`	`DifferentRoom`
`Overlap`	

Table 7.3: Distribution constraints providing edges to the class-room conflict graph.

et al. (2018). Thus, it is easy to check if there is an edge between two vertices (a pair of $w_{c,r}$).

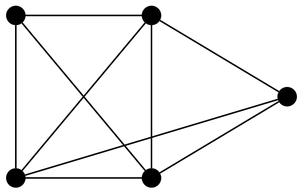
`Overlap`: Does not explicitly depend on the room assignments, but for a pair of classes that must be overlapping, they cannot be assigned the same room since it would violate the room double booking constraint.

7.4.3 Class-time-room conflict graphs

A class-time-room conflict graph for an entire problem data set is usually huge in the number of edges, making it very time and memory consuming to use. When used in this paper, they model smaller parts of the problem, such as time-room overlaps for the `SameAttendees` distribution constraints and student conflicts.

7.4.4 Clique cover

We use cliques in the hard constraint conflict graphs. A clique is a subset of vertices where every pair of vertices is adjacent. Thus, a clique states that, at most, one of the vertices can be used in a feasible solution. Covering the hard constraint conflict graph with the largest cliques possible will then give the tightest constraints. The problem of finding a maximal clique cover is $\mathcal{NP}$-complete. Bron and Kerbosch, 1973 provides an algorithm for finding all maximal cliques in a graph; however, the Bron-Kerbosch algorithm is too time-consuming on our graphs. Instead, we use the Kellerman algorithm as described in Gramm et al., 2006. Figure 7.1 shows an example of a clique cover.


(a) A hard constraint conflict graph.

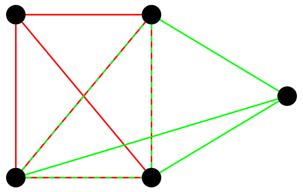

(b) Two cliques that covers all edges. The three dashed edges are covered by both cliques.

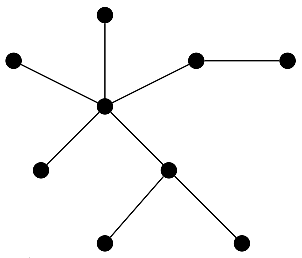
Figure 7.1: An example of a hard constraint conflict graph (7.1a) and a clique cover (7.1b).

7.4.5 Star cover

When we want to cover a hard constraint class-time-room conflict graph, it is often not possible to use a clique cover since the graph contains too many edges. We introduce a star cover in such situations. When we use the star cover, we do not include edges between equal-class vertices, making the graph more sparse. A star is a tree with one internal vertex and a number of leaves. The star translates into a constraint where either the internal vertex is used or any number of the leaves. For any star, we have that the internal vertex is of class c_i, and all leaves are of class $c_j \neq c_i$. A star cover is found by iteratively finding the largest star (the vertex with the highest degree of unlabeled edges) and labeling the edges until all edges are labeled. The pseudocode is shown in Algorithm 2. Figure 7.2 shows an example of a star cover.

Algorithm 2 Create star cover

```
function StarCover($G$)
    S = list of stars, initially empty
    V = list of vertices ordered by descending degree
    while V contains unlabeled vertices do
        v = first unlabeled vertex of v
        N = neighbours of v with unlabeled edge
        s = star with internal vertex v and leaves N
        add s to S
        label edges v – n
        label v
        reorder V
    end while
    return S
end function
```


(a) A hard constraint conflict graph.

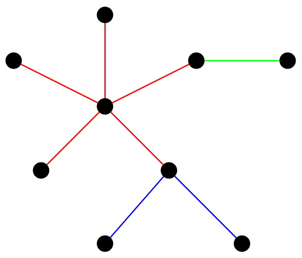

(b) Three stars that covers all edges.

Figure 7.2: An example of a hard constraint conflict graph (7.2a) and a star cover (7.2b).

7.4.6 Special star cover

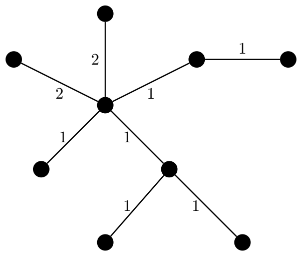
A soft constraint conflict graph does not, by definition, include edges between equal-class vertices; therefore, we use stars to cover such graphs. A special star is a star where all edges have the same weight. We find the special star cover by splitting the graph G into subgraphs G' where all edges have equal weight and all leaves are of the same class. For each subgraph, G', the special stars are found by iteratively taking the largest star and splitting it into special stars by labeling the edges according to the leaf's class. The process terminates when all edges are labeled. Algorithm 3 and 4 shows the procedures. The special star allows for either the internal vertex or a leaf to be used without conflict. If both are used, there will be a conflict with a penalty equal to the edge cost. Figure 7.3 shows an example of a special star cover.

Algorithm 3 Create special star cover

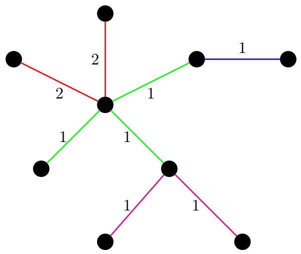
```
function SPECIALSTARCOVER($G$)
    S = list of stars, initially empty
    split $G$ into subgraphs $G'$ of equal edge weight
    for all $G'$ in $G$ do
        s = SPECIALSTARCOVERSUBGRAPH($G'$)
        add s to S
    end for
    return S
end function
```

Algorithm 4 Create special star cover for subgraph

```
function SPECIALSTARCOVERSUBGRAPH($G'$)
    S = list of stars, initially empty
    V = list of vertices ordered by descending degree
    while V contains unlabeled vertices do
        v = first unlabeled vertex of v
        N = neighbours of v with unlabeled edge
        Ng = neighbours grouped by class
        for all n in Ng do
            s = star with internal vertex v and leaves n
            add s to S
            label edges between v and n
        end for
        label v
        reorder V
    end while
    return S
end function
```



(a) A soft constraint conflict graph.



(b) Four stars that covers the graph on Figure 7.3a. Note that all stars have equal weight on the edges.

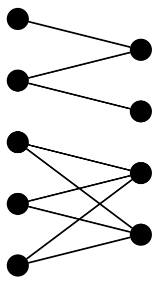
Figure 7.3: An example of a soft constraint conflict graph with different edge weights (7.3a) and the respective special star cover (7.3b).

7.4.7 Complete bipartite cover

A bipartite graph is a graph whose vertices can be divided into two disjoint subsets $\mathcal{V}_1$ and $\mathcal{V}_2$, such that all edges connect a vertex in $\mathcal{V}_1$ with a vertex in $\mathcal{V}_2$. A complete bipartite graph contains all possible edges such that all vertices of $\mathcal{V}_1$ are connected to all vertices of $\mathcal{V}_2$.

Consider an overlap-conflict graph that contains only vertices of two different classes, which is the case when modeling the overlap of two classes used in the student conflict constraints. The graph does not contain edges between equal-class vertices since an overlap can only occur between classes. Naturally, that would lead to the use of a star cover. In this case, we have only vertices of two different classes, which means that we have a bipartite graph. Consider a star cover of a bipartite graph and two stars with the same leaves. A star in a bipartite graph is, by nature, a complete bipartite subgraph. Combining two stars with the same leaves form a larger complete bipartite subgraph. Each complete bipartite subgraph creates a constraint that says that only one of the vertices can be used without generating an overlap.

We cover the bipartite conflict graph by large complete bipartite subgraphs and denote this as a complete bipartite cover. We do so by selecting the largest of the sets $\mathcal{V}_1$ and $\mathcal{V}_2$, and labeling two vertices the same if they have exactly the same neighbors. For each complete bipartite subgraph, we can use at most one vertex without generating a conflict. Since we cannot use vertices from the same subset, a second vertex must be from the other subset, and as the graph is complete, there must exist an edge between the two vertices, thus creating a conflict. Figure 7.4 shows an example of a complete bipartite cover.


(a) A hard conflict graph.

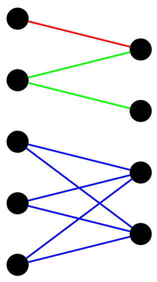
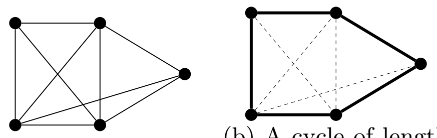

(b) Three complete bipartite graphs that covers all edges.

Figure 7.4: An example of a hard conflict graph (7.4a) and a complete bipartite cover (7.4b).

7.4.8 Odd cycles

We use odd cycles in the hard constraint conflict graph. A cycle is a sequence of distinct edges joining a set of vertices such that only the first and last vertex is repeated. An odd cycle is a cycle with an odd number of vertices. In general, for odd cycles, we can use at most $\frac{n-1}{2}$ of the vertices in a solution, where n is the number of vertices in the cycle. If the odd cycle contains more than $\frac{n}{2}$ distinct classes, we can not use it to strengthen the model. We search the graph for odd cycles by finding cycles in a depth first search. We then remove all cycles with even n and cycles that are part of a clique since these do not strengthen the formulation. We do not use the odd cycles as a cover, but we create a set of valid inequalities from the found odd cycles. Figure 7.5 shows an example of an odd cycle.



(a) A conflict graph. (b) A cycle of length 5.

Figure 7.5: An example of an odd cycle not contained in any clique.

7.5 Preprocessing the data instances

The problem with having unnecessary constraints and variables in the input data is that it generates many unneeded operations that consume time and memory to be evaluated but do not provide any essential information. This means that time is spent on valueless operations instead of solving the problem. The time saved can be seen in different steps. When generating the graphs, it will be faster the fewer vertices there exists. Also, fewer constraints mean fewer class pair checks to know if an edge should be added. With fewer vertices, there will also be fewer edges; thus, it will be faster to search through the graphs. When a constraint considers an unnecessary variable, it constrains a case that is not feasible, resulting in generating more constraints or adding more variables in summations than needed. Note that when removing, for example, an available time from a class, the reduction in variables is multiplied with the number of available rooms.

By reducing the number of constraints and variables before giving the model to a commercial solver, the commercial solver will spend less time presolving and perhaps result in a stronger model as the input model is tighter than otherwise. By using the knowledge about the problem structure, it is faster to do the presolving oneself than letting the solver identify reductions from a large MIP model[2]. In general, fewer variables/constraints lead to faster solving time and faster branching.

[2]hundreds of thousands or millions of variables and constraints

7.5.1 Reducing the number of distribution constraints

We consider two types of reduction of the distribution constraints; *redundant* and *dominated* distributions constraints.

7.5.1.1 Redundant constraints

Redundant distribution constraints are constraints that can be removed without changing the feasible region of the problem. The identification of redundant constraints is described by the following:

Single classes: The number of distinct classes is less than or equal to 1. We remove the distribution.

Zero penalty: The distribution constraint is soft, and the penalty is 0. We remove the distribution.

No violation: The set of eligible times or rooms of the classes that the constraint considers cannot violate the given constraint. We check this by creating a class-time or class-room conflict graph of the given distribution constraint. If the conflict graph contains no edges, then we remove the distribution constraint. We check this for each of the distribution constraint types of Table 7.1 and Table 7.3.

7.5.1.2 Dominated constraints

The identification of dominated constraints is described by the following:

Dominated constraints: A hard or soft distribution constraint d_i is dominated by a hard distribution constraint d_j if they are of the same type and the classes of d_i is a subset of the classes of d_j.

Dominated constraints special case: A hard or soft distribution constraint d_i of type `NotOverlap` is dominated by a hard distribution constraint d_j, if d_j is of type `SameAttendees` and the classes of d_i is a subset of the classes of d_j. This is true because the `SameAttendees` disallows for classes to be assigned overlapping times.

7.5.2 Reducing the number of variables

By performing operations on the conflict graphs, we reduce the number of variables. We use the hard constraint class-time and class-room conflict graphs for this reduction. By using the fact that each class must be scheduled (one vertex from each class must be used), we reduce the number of vertices in the graph in two ways:

Reduction by fixed vertices: Some classes have only one eligible time and/or room, which means that it has only one vertex in the graph. We denote such a vertex as *fixed*. We can never use a neighbor of a fixed vertex in a feasible solution; thus, we remove the neighbor (and edges) from the graph. The algorithm to reduce neighbors of fixed vertices is shown in Algorithm 5

Reduction by cliques: Consider a clique that contains all vertices of class c_i and includes vertices of other classes. The clique represents that only one of the vertices can be used in a feasible solution. We know that this vertex must be of class c_i since c_i does not have any other vertex outside the clique. This means that we cannot use the clique-vertices that are not of class c_i in a feasible solution, and thus, we remove these vertices and their edges from the graph. The clique-reduction algorithm is shown in Algorithm 5

Since our reduction methods remove vertices and edges from the graph, the reductions might lead to new fixed vertices. If a class has all vertices but one removed, the remaining vertex must be fixed. Thus the graph reduction should be performed by alternating between the two methods until no further reductions can be made. The overall reduction algorithm is shown in Algorithm 7.

Algorithm 5 Reduction by fixed vertices

```
1: procedure ReduceFixed($G$)
2:     Identify fixed vertices
3:     n = list of neighbors of fixed vertices
4:     while n is not empty do
5:         Remove n from graph
6:         Identify fixed vertices
7:         n = list of neighbors of fixed vertices
8:     end while
9: end procedure
```

Algorithm 6 Reduction by cliques

```
1: procedure ReduceClique($G$)
2:     $\Omega_G$ = a clique cover
3:     for all $\omega \in \Omega_G$ do
4:         if $\omega$ contains all vertices of a class $c_i$ then
5:             Remove vertices $v \in \{\omega : v_c \neq c_i\}$
6:         end if
7:     end for
8: end procedure
```

Algorithm 7 Graph reduction algorithm

```
1: procedure ReduceClique($G$)
2:     ReduceFixed($G$)
3:     while Reductions are made do
4:         ReduceClique($G$)
5:         ReduceFixed($G$)
6:     end while
7: end procedure
```

7.6 Graph-based distribution constraints

We use the conflict graphs presented in Section 7.4 to model most of the distribution constraints. Table 7.4 gives an overview of the modeling. This section provides a description of the clique and star constraints and an examination of the special distribution constraint which are partly included in the conflict graphs. For any soft constraint, we will use the p as the penalty variable. To simplify the constraints p is presented dimensionless.

Constraint	Hard	Soft
`SameStart`	Clique	Star
`SameTime`	Clique	Star
`SameDays`	Clique	Star
`SameWeeks`	Clique	Star
`SameRoom`	H2020	H2020
`DifferentTime`	Clique	Star
`DifferentDays`	Clique	Star
`DifferentWeeks`	Clique	Star
`DifferentRoom`	H2020	H2020
`NotOverlap`	Clique	Star
`Overlap`	Clique	Star
`SameAttendees`	Clique and star	Star
`Precedence`	Clique	Star
`Workday(S)`	Clique	Star
`MinGap(G)`	Clique	Star
`MaxDays(D)`	Clique and H2020	H2020
`MaxDayLoad(S)`	Clique and H2020	H2020
`MaxBreaks(R,S)`	H2020	H2020
`MaxBlock(M,S)`	Clique and H2020	H2020

Table 7.4: The modeling methods of the different distribution constraint types. H2020 indicates that the formulation by Holm et al. (2020) is used.

7.6.1 Clique constraints

The hard constraint class-time conflict graph described in Section 7.4.1.1 is used to generate clique constraints (Nemhauser and Wolsey, 2014). The clique cover is used to model all hard constraints of the types described in Table 7.1. A clique $\omega \in \Omega_G$ is a set of vertices, where each vertex v represents a time-assignment of a class $v = \{v_c, v_t\}$, where $v_c \in \mathcal{C}$ and $v_t \in \mathcal{T}_c$, i.e. the $y_{c,t}$ variables.

$$\sum_{v \in \omega} y_{v_c,v_t} \leq 1 \qquad \forall\, \omega \in \Omega_G$$

7.6.2 Star constraints

The soft class-time conflict graph G described in Section 7.4.1.2 is used to model the soft distribution constraints of the types shown in Table 7.1. The constraints are modeled using a special star cover $\mathcal{S}_G$ of the graph G as described in Section 7.4.6. A star $s \in \mathcal{S}_G$ represents an internal vertex v^{in} and a set of leaves $\mathcal{V}_l(s)$. The penalty variable is binary $p \in \{0,1\}$ and since each star s have equal costs on the edges ζ_e the costs of the penalty variable is $c_p = \zeta_e$.

$$y_{v_c^{\text{in}},v_t^{\text{in}}} + \sum_{v^l \in \mathcal{V}_l(s)} y_{v_c^l,v_t^l} - 1 \leq p \qquad \forall s \in \mathcal{S}_G$$

7.6.3 Odd-cycle constraints

Odd-cycle constraints are valid inequalities(Nemhauser and Wolsey, 2014). We find the odd cycles in the hard constraint class-time conflict graph as described in Section 7.4.8. The set of odd cycles is denoted $\mathcal{O}_G$, where $o \in \mathcal{O}_G$ is a set of vertices $o \subset \mathcal{V}$ and $|o|$ is the cardinality of o. We have the odd-cycle constraints

$$\sum_{v \in o} y_{v_c,v_t} \leq \frac{|o|-1}{2} \qquad \forall\, o \in \mathcal{O}_G$$

7.6.4 `SameRoom` - `DifferentRoom`

The `SameRoom` constraint is modeled by comparing the classes pairwise. The constraints restrict that if one class uses r_i then the other class cannot use room $r_{j \neq i}$.

$$\text{Hard:} \quad w_{c_i,r_i} + \sum_{r_j \in \mathcal{R}_{c_j} \setminus \{r_i\}} w_{c_j,r_j} \leq 1$$

$$\text{Soft:} \quad w_{c_i,r_i} + \sum_{r_j \in \mathcal{R}_{c_j} \setminus \{r_i\}} w_{c_j,r_j} - 1 \leq p$$

The `DifferentRoom` constraints are modeled with the same idea, but with an opposite logical expression as the constraint is opposite.

$$\text{Hard:} \qquad w_{c_i,r} + w_{c_j,r} \leq 1$$

$$\text{Soft:} \qquad w_{c_i,r} + w_{c_j,r} - 1 \leq p$$

For more details on the constraints, we refer to Holm et al., 2020.

7.6.5 `SameAttendees`

The `SameAttendees` constraint says that two classes cannot overlap in time or in a time-room combination. The overlap in time is covered by the clique constraints (hard) or the star constraints (soft) of the class-time conflict graphs. Thus this section considers the time-room overlap only.
We model the time-room overlap using a class-time-room conflict graph G. An edge is then added between two vertices if there is a time-room overlap between the vertices. We do not add edges if the vertices represent the same class or if the times overlap. We model the hard constraints by a star cover and the soft constraints by a special star cover. A star $s \in \mathcal{S}_G$ has an internal vertex v^{in} and a set of leaves $\mathcal{V}_l(s)$. M is the number of different classes represented in the leaf vertices; $M = \left|\cup_{v \in \mathcal{V}_l(s)} v_c\right|$. The penalty variable is binary; $p \in \{0,1\}$ and $c_p = c_\delta$.

$$\text{Hard:} \qquad M x_{v_c^{\text{in}}, v_t^{\text{in}}, v_r^{\text{in}}} + \sum_{v^l \in \mathcal{V}_l(s)} x_{v_c^l, v_t^l, v_r^l} \leq M \qquad \forall s \in \mathcal{S}_G$$

$$\text{Soft:} \qquad x_{v_c^{\text{in}}, v_t^{\text{in}}, v_r^{\text{in}}} + \sum_{v^l \in \mathcal{V}_l(s)} x_{v_c^l, v_t^l, v_r^l} - 1 \leq p \qquad \forall s \in \mathcal{S}_G$$

7.6.6 Hard `MaxDays(D)`

Recall that the hard constraint class-time conflict graph contains edges from the `MaxDays(D)` distributions constraints. This means that if a violation between two classes exists, the clique constraints cover it. It also means that if one of the classes in the distribution constraint must be scheduled on `D` days, then all other classes must use a subset of those `D` days, and thus the edges in the graph are sufficient to model the constraint and no further constraints are added. All other possible violations are modeled by the constraints shown below (Holm et al., 2020).

$$\sum_{d \in \mathcal{D}} \gamma_d \leq \mathtt{D}$$

Where

$$\gamma_d = \begin{cases} 1 & \text{if any class } c \in \mathcal{C}_\delta \text{ is scheduled on day } d \\ 0 & \text{otherwise} \end{cases}$$

7.6.7 Soft `MaxDays(D)`

There is no graph that includes edges of the soft
`MaxDays(D)` constraints. The constraints are modeled as (Holm et al., 2020)

$$\sum_{d\in\mathcal{D}} \gamma_d - \mathtt{D} \leq p$$

7.6.8 Hard `MaxDayload(S)`

The hard constraint class-time conflict graph contains all the possible violations when considering a pair of classes. This implies that if the number of classes scheduled in week w and day d is at most 2, then the clique constraints are sufficient for that week and day. In all other cases, we have the day load $\phi_{w,d} \in \mathbb{Z}^+$ on a day d in a week w and

$$\phi_{w,d} = \sum_{\substack{c\in\mathcal{C}_\delta,\\ t\in\mathcal{T}_c:t^{\text{weeks}}=w\wedge t^{\text{days}}=d}} t^{\text{length}} y_{c,t} \qquad \forall w\in\mathcal{W}, d\in\mathcal{D}$$

Thus the constraints are as follows.

$$\phi_{w,d} \leq \mathtt{S} \qquad \forall w\in\mathcal{W}, d\in\mathcal{D}$$

7.6.9 Soft `MaxDayload(S)`

There is no graph that includes edges of the soft
`MaxDayLoad(S)` constraints. The constraints are modeled as (Holm et al., 2020)

$$\phi_{w,d} - \mathtt{S} \leq \iota_{w,d} \qquad \forall w\in\mathcal{W}, d\in\mathcal{D}$$

The variable $\iota_{w,d} \in \mathbb{Z}^+$ counts the number of exceeding time slots on day d in week w. Thus the penalty for this distribution constraint is set by (Holm et al., 2020):

$$\frac{c_\delta}{|\mathcal{W}|} \sum_{\substack{w\in\mathcal{W},\\ d\in\mathcal{D}}} \iota_{w,d} - 0.999 \leq p$$

7.6.10 Hard `MaxBlock(M,S)`

Holm et al. (2020) defined special cases when the length of a class was longer than `M`. These *long classes* can be scheduled alone but cannot be scheduled such that they are in a block with another class from the constraint. This special case is included in the class-time conflict graph and is thus modeled by the clique constraints. Therefore, we do not need such special cases. For regular cases, we have the binary variable $\rho_{w,d,\tau}$ that is set to 1 if a block longer than `M` starts in week w on day d at time slot τ (Holm et al., 2020).

$$\sum_{\substack{w\in\mathcal{W},\\ d\in\mathcal{D},\\ \tau\in\mathrm{T}}} \rho_{w,d,\tau} = 0$$

7.6.11 Soft `MaxBlock(M,S)`

The soft `MaxBlock(M,S)` is modeled like the hard but with an associated penalty $p \in \mathbb{Z}^+$ (Holm et al., 2020).

$$\frac{c_\delta}{|\mathcal{W}|} \sum_{\substack{w \in \mathcal{W}, \\ d \in \mathcal{D}, \\ \tau \in \mathrm{T}}} \rho_{w,d,\tau} \leq p$$

7.7 Student sectioning

We have identified various structures in the student sectioning where we can take advantage of some of the implicit properties, and in combination with using conflict graphs, we reduce the number of variables and constraints used in student sectioning.
We reduce the number of student sectioning constraints by identifying classes that must be attended by the same students. We identify such class pairs before creating the student conflict constraints. In the following, these class pairs are described.

7.7.1 Inevitable conflicts

There exists class pairs that have a structure such that no matter the student sectioning, an overlap within these class pairs will result in a fixed number of student conflicts. Thus we can construct a conflict graph with the fixed number of student conflicts as the weight on the edges. Since we have class pairs, we have a bipartite graph and can model this from a complete bipartite cover, as described in Section 7.4.7. We can use the inevitable conflicts to penalize in the objective function solely on the overlap between pairs of classes, no matter the student sectioning.

7.7.1.1 Courses with mandatory classes

Consider a course with only one configuration. If there exists a subpart with only one class (with a limit greater than zero), the class is mandatory for all students attending the course. Denote the set of mandatory classes $\mathcal{C}^{\mathrm{m}}$.

$$e_{s,c} = 1 \qquad \forall\, c \in \mathcal{C}^{\mathrm{m}}, s \in \mathcal{S}_c$$

Any class pair $(c_i, c_j) \in \mathcal{C}^{\mathrm{m}}$ that has students in common will generate a fixed number of student conflicts if the classes overlap. The number of student conflicts generated from an overlap will be the number of students that the classes have in common $\left|\mathcal{S}_{c_i} \cap \mathcal{S}_{c_j}\right|$, which show in the objective.

7.7.1.2 Full subparts

Consider a course that has only one configuration, and one or more subparts are being filled, such that all classes in the subpart have their maximum number of

attending students. If a class c_i is full and has a parent c_i^{parent}, then the same students must attend the parent and thus create c_i^{limit} conflicts if c_i and c_i^{parent} overlaps.
Also, if there exists another subpart that has only one class c_j, then since the students of the full class c_i must attend a class from each subpart of the configuration, they must also attend class c_j and thus create c_i^{limit} conflicts if c_i and c_j overlaps.
Both cases generate c_i^{limit} conflicts if the classes overlaps, which shows in the objective.

7.7.2 Impossible conflicts

The number of student sectioning constraints can be reduced by identifying pairs of classes that by definition cannot have student conflicts. That can be class pairs (c_i, c_j) with one of the following properties:

1. (c_i, c_j) are in a hard `SameAttendees` distribution constraint
2. (c_i, c_j) are from different courses and no student must attend both courses
3. (c_i, c_j) are from the same course and
   1. from the same subpart or
   2. from different configurations or
   3. c_i^{parent} is from the same subpart as c_j and $c_i^{\text{parent}} \neq c_j$

For all the cases in the above list we know that there cannot be any student conflicts because the classes cannot conflict (case 1) and/or no student can attend both classes (case 2 and 3), thus we do not need to generate any student sectioning constraints or variables.

7.7.3 Overlapping classes

For the remaining class pairs (not covered by Section 7.7.1 or skipped by Section 7.7.2) we split the overlap into two types; the *time overlap* and the *time-room overlap*. Time overlap happens when the assigned times overlap, and the time-room overlap happens when the assigned times do not overlap themselves, but the distance between the assigned rooms create an overlap. The classes overlap if either overlap type exists. We have the overlapping variable

$$o_{c_i,c_j} = \begin{cases} 1 & \text{if classes } c_i \text{ and } c_j \text{ overlaps} \\ 0 & \text{otherwise} \end{cases}$$

7.7.3.1 Time overlap

If there exists a hard `NotOverlap` distribution constraint between the two classes, then a time overlap cannot exist, and this time overlap constraint is skipped. If not, then a class-time conflict graph is generated for the two classes only. The conflict graph has an edge between two vertices of different classes if the times are overlapping. The graph is thus bipartite, and the constraints are defined by a complete bipartite graph cover $\mathcal{B}_G$ (see Section 7.4.7). A complete bipartite graph $b \in \mathcal{B}_G$ is a set of vertices, where each vertex v represents a time-assignment of a class $v = \{v_c, v_t\}$, i.e. the $y_{c,t}$ variables.

$$\sum_{v \in b} y_{v_c,v_t} - 1 \leq o_{c_i,c_j} \qquad \forall\, b \in \mathcal{B}_G$$

7.7.3.2 Time-room overlap

We model he time-room overlap in a similar way to the time overlap. We construct a class-time-room conflict graph, where there exists an edge between two vertices if they are of different classes, and the times do not overlap, but the room assignments result in an overlap. Again a complete bipartite graph cover $\mathcal{B}_G$ is used to model the constraints.

$$\sum_{v \in b} x_{v_c,v_t,v_r} - 1 \leq o_{c_i,c_j} \qquad \forall\, b \in \mathcal{B}_G$$

Note that if neither the class-time conflict graph nor the class-time-room conflict graph contains any edges, then the two classes cannot overlap and o_{c_i,c_j} can be fixed to zero (and removed from the model).
We define the set $\mathcal{C}'$ as the set of pairs $\{(c_i, c_j) : i < j\}$ that are not covered by Section 7.7.1 and are not skipped by Section 7.7.2, i.e. they do not satisfy any of the cases 1-3 and their conflict graphs contains at least one edge. Thus we have the constraint

$$o_{c_i,c_j} + e_{s,c_i} + e_{s,c_j} - 2 \leq \chi_{s,c_i,c_j} \quad \forall\, (c_i, c_j) \in \mathcal{C}', s \in \mathcal{S}_{c_i} \cap \mathcal{S}_{c_j}$$

where $e_{s,c}$ is the decision variable defined in Section 7.3 and χ_{s,c_i,c_j} denotes if there exists a student conflict between student s and classes c_i and c_j.

7.8 Objective

There are four categories of objectives: time, room, distribution constraints, and student conflicts. These categories have respective weights ψ_t, ψ_r, ψ_δ, and ψ_s that prioritize the types of objectives compared to each other. We use the same objectives defined by Holm et al. (2020) for the *time*, *room* and *distribution constraints*. But the *student conflicts* objective is split into three parts; the two cases from Section 7.7.1 and the 'regular' case from Section 7.7.3 Each time assigned to a class is penalized with a penalty $p_{c,t} \geq 0$ and an overall objective weight ψ_t.

$$\psi_t \sum_{\substack{c \in \mathcal{C}, \\ t \in \mathcal{T}_c}} p_{c,t} y_{c,t}$$

Each room assigned to a class is penalized with a penalty $p_{c,r} \geq 0$ and an overall objective weight ψ_r.

$$\psi_r \sum_{\substack{c \in \mathcal{C}, \\ r \in \mathcal{R}_c}} p_{c,r} w_{c,r}$$

Each soft distribution constraint defined a penalty variable p and a penalty cost $c_p > 0$. We denote the set of all penalty variables $\mathcal{P}$.

$$\psi_\delta \sum_{p \in \mathcal{P}} c_p p$$

The student conflicts are are penalized according to the description of Section 7.7. The mandatory class pairs $\mathcal{C}^{\mathrm{m}}$ and full subpart class pairs $\mathcal{C}^{\mathrm{f}}$ are penalized by overlap only as described in Section 7.7.1.1 and 7.7.1.2. The remaining class pairs are penalized by the student conflict variable χ_{s,c_i,c_j}. The overall weight of the student sectioning is denoted by ψ_s.

$$\psi_s \Big(\sum_{(c_i,c_j) \in \mathcal{C}^{\mathrm{m}}} \left| \mathcal{S}_{c_i} \cap \mathcal{S}_{c_j} \right| o_{c_i,c_j} + \sum_{(c_i,c_j) \in \mathcal{C}^{\mathrm{f}}} c_i^{\mathrm{limit}} o_{c_i,c_j} + \sum_{\substack{(c_i,c_j) \in \mathcal{C}', \\ s \in \mathcal{S}_{c_i} \cap \mathcal{S}_{c_j}}} \chi_{s,c_i,c_j} \Big)$$

The objective function is a combined sum over the four objective parts described above.

7.9 Results

The results are produced on a 64bit computer running Scientific Linux 7.7. The machine is equipped with two Intel Xeon E5-2650 v4 CPUs clocked at 2.20GHz and 256GB of RAM. We use the 30 instances from the ITC 2019 competition as test data. The MIP is solved using Gurobi 9.0 with 8 threads.
There is no MIP-related data available for instance *pu-proj-fal19* as we could not construct the MIP model because of a lack of memory, which leads to blank space in Table 7.6, 7.7, 7.8, 7.9, 7.13, 7.14, and 7.15.

The distribution constraints of each data instance is reduced according to the description in Section 7.5.1. The total number of reduced distribution constraints is given in Table 7.5. The specified number of reductions for each type is given in Appendix 7.B.
The variables have been reduced as presented in Section 7.5.2. Table 7.6 shows the total number of removed vertices and the number of vertices removed from the class-room and class-time graphs, respectively. The table also shows the percentage of $x_{c,t,r}$ variables that can be eliminated from the MIP. Appendix 7.C shows the number of vertices removed by fixed vertices and by cliques.

Instance	Removed	Total	Percentage
agh-fis-spr17	44	1,223	3.60 %
agh-ggis-spr17	118	2,690	4.39 %
bet-fal17	92	1,251	7.35 %
iku-fal17	245	2,903	8.44 %
mary-spr17	72	3,951	1.82 %
muni-fi-spr16	74	740	10.00 %
muni-fsps-spr17	79	400	19.75 %
muni-pdf-spr16c	231	2,028	11.39 %
pu-llr-spr17	118	645	18.29 %
tg-fal17	301	503	59.84 %
agh-ggos-spr17	75	1,689	4.44 %
agh-h-spr17	27	399	6.77 %
lums-spr18	28	518	5.41 %
muni-fi-spr17	96	710	13.52 %
muni-fsps-spr17c	95	709	13.40 %
muni-pdf-spr16	267	1,012	26.38 %
nbi-spr18	8	596	1.34 %
pu-d5-spr17	211	1,535	13.75 %
pu-proj-fal19	805	7,837	10.27 %
yach-fal17	58	645	8.99 %
agh-fal17	295	7,158	4.12 %
bet-spr18	110	1,418	7.76 %
iku-spr18	308	3,488	8.83 %
lums-fal17	51	599	8.51 %
mary-fal18	99	515	19.22 %
muni-fi-fal17	87	798	10.90 %
muni-fspsx-fal17	182	1,361	13.37 %
muni-pdfx-fal17	628	3,501	17.94 %
pu-d9-fal19	413	2,755	14.99 %
tg-spr18	236	426	55.40 %
Average			13.67 %

Table 7.5: Number of reductions of distribution constraints.

7.9.1 Solution results

Table 7.7 gives an overview of the number of constraints and variables for the graph-based MIP formulation presented in this paper and the basic MIP formulation presented by Holm et al. (2020). Many of the instances are reduced significantly regarding the number of constraints. The effects of strengthening differ more when considering the number of variables. Two instances even had an increase in the number of variables; this is due to an increased number of auxiliary variables used to model the student conflicts. The reason for an increased number of student conflict auxiliary variables is that the number of auxiliary variables used to model the constraints of Section 7.7.1 is larger than the number auxiliary variables saved from not modeling the pairs as in Section

Instance	Reduced vertices in total	Reduced class -room vertices	Reduced class -time vertices	Reduction of $x_{c,t,r}$
agh-fis-spr17	11,167	12	11,155	9.05 %
agh-ggis-spr17	3,656	46	3,610	8.02 %
bet-fal17	369	0	369	1.83 %
iku-fal17	8,616	7	8,609	8.46 %
mary-spr17	117	0	117	0.77 %
muni-fi-spr16	878	0	878	10.17 %
muni-fsps-spr17	650	0	650	5.87 %
muni-pdf-spr16c	24,247	1	24,246	15.68 %
pu-llr-spr17	38	9	29	0.37 %
tg-fal17	4,997	0	4,997	20.76 %
agh-ggos-spr17	6,458	42	6,416	3.48 %
agh-h-spr17	832	38	794	2.51 %
lums-spr18	625	0	625	3.08 %
muni-fi-spr17	506	0	506	5.78 %
muni-fsps-spr17c	5,131	0	5,131	8.22 %
muni-pdf-spr16	1,185	93	1,092	2.74 %
nbi-spr18	721	0	721	1.89 %
pu-d5-spr17	746	1	745	5.58 %
pu-proj-fal19	3,539	57	3,482	
yach-fal17	604	0	604	3.89 %
agh-fal17	15,973	273	15,700	4.50 %
bet-spr18	548	0	548	2.03 %
iku-spr18	15,891	229	15,662	16.68 %
lums-fal17	340	0	340	1.75 %
mary-fal18	258	0	258	2.52 %
muni-fi-fal17	297	6	291	3.07 %
muni-fspsx-fal17	6,364	3	6,361	7.20 %
muni-pdfx-fal17	30,719	106	30,613	13.81 %
pu-d9-fal19	1,310	101	1,209	4.55 %
tg-spr18	5,893	0	5,893	39.45 %
Average				7.37 %

Table 7.6: Overview of the reduction performed by fixed vertices and cliques.

	Basic MIP		Graph-based MIP		Reduction	
Instance	Constraints	Variables	Constraints	Variables	Constraints	Variables
agh-fis-spr17	10,716,234	5,270,957	1,806,658	2,159,378	83.14 %	59.03 %
agh-ggis-spr17	10,971,106	10,532,658	1,330,393	1,472,214	87.87 %	86.02 %
bet-fal17	7,427,922	4,770,806	2,974,988	2,426,521	59.95 %	49.14 %
iku-fal17	11,440,488	3,772,222	296,727	3,449,418	97.41 %	8.56 %
mary-spr17	1,716,685	1,287,460	423,901	402,043	75.31 %	68.77 %
muni-fi-spr16	4,958,093	4,350,531	1,970,020	1,636,839	60.27 %	62.38 %
muni-fsps-spr17	1,003,857	717,147	330,911	275,897	67.04 %	61.53 %
muni-pdf-spr16c	26,203,626	6,922,578	3,527,034	4,338,790	86.54 %	37.32 %
pu-llr-spr17	8,980,995	5,542,707	1,671,963	1,344,501	81.38 %	75.74 %
tg-fal17	597,723	232,749	52,556	82,223	91.21 %	64.67 %
agh-ggos-spr17	15,912,008	3,523,533	2,458,347	2,304,596	84.55 %	34.59 %
agh-h-spr17	12,642,901	1,881,442	1,623,559	1,674,303	87.16 %	11.01 %
lums-spr18	589,187	458,344	121,663	442,590	79.35 %	3.44 %
muni-fi-spr17	6,475,267	5,711,500	2,783,030	2,299,293	57.02 %	59.74 %
muni-fsps-spr17c	17,717,840	1,466,645	3,530,470	2,077,599	80.07 %	-41.66 %
muni-pdf-spr16	10,554,793	6,901,019	4,264,196	3,423,148	59.60 %	50.40 %
nbi-spr18	1,955,308	631,823	549,940	592,387	71.87 %	6.24 %
pu-d5-spr17	31,570,758	30,624,014	4,300,705	4,002,079	86.38 %	86.93 %
pu-proj-fal19						
yach-fal17	4,407,666	1,336,902	1,226,721	659,118	72.17 %	50.70 %
agh-fal17	45,591,515	25,795,085	10,071,748	9,788,065	77.91 %	62.05 %
bet-spr18	10,006,170	6,165,343	4,070,348	3,022,369	59.32 %	50.98 %
iku-spr18	9,617,259	3,728,533	276,717	3,074,521	97.12 %	17.54 %
lums-fal17	525,567	448,222	106,246	437,140	79.78 %	2.47 %
mary-fal18	3,754,624	3,307,703	1,355,615	1,272,392	63.89 %	61.53 %
muni-fi-fal17	8,032,513	7,384,314	3,391,653	2,893,880	57.78 %	60.81 %
muni-fspsx-fal17	23,884,170	2,791,093	5,708,241	3,253,105	76.10 %	-16.55 %
muni-pdfx-fal17	48,957,509	22,345,083	7,135,695	8,322,330	85.42 %	62.76 %
pu-d9-fal19	115,664,227	56,683,344	18,878,893	11,297,549	83.68 %	80.07 %
tg-spr18	645,009	130,533	72,087	91,937	88.82 %	29.57 %
Average					77.18 %	42.96 %

Table 7.7: Size of basic MIP formulation by Holm et al. (2020) compared with the graph-based formulation.

7.7.3. This is the case for some of the instances but only visible in the overall reduction of instances *muni-fsps-spr17c* and *muni-fspsx-fal17*.
Modern commercial solvers provide a palette of presolve methods which aims to reduce and strengthen the model that it is given. To compare the two formulations it is also sensible to compare the problem sizes after the presolve methods by a commercial solver, in this case Gurobi 9.0.

The presolve stats of Table 7.8 shows that the graph-based model is not always the smallest when we compare the number of variables. On the other hand, the graph-based model always contains fewer constraints and causes a faster presolve procedure. We should also note that the presolve procedure is not deterministic, and our stats are for a single run. To see the effects of the strengthening, we provide a comparison of the 1 hour and 24 hour runtime stats.

Table 7.9 shows the runtime stats for the graph-based MIP compared to the basic MIP for 1 hour and 24 hours, respectively. The results show that the graph-based MIP finds lower and upper bounds to more instances than the basic MIP, the bounds are mostly as good as the ones found by the basic MIP.

	Basic MIP			Graph-based MIP			Reduction		
Instance	Constraints	Variables	Time(sec)	Constraints	Variables	Time(sec)	Constraints	Variables	Time(sec)
agh-fis-spr17	7,412,351	2,582,019	2,956	1,671,689	2,027,595	966	77.45 %	21.47 %	-1,989
agh-ggis-spr17	1,966,325	1,961,118	239	1,221,091	1,317,979	87	37.90 %	32.79 %	-152
bet-fal17	6,219,867	3,982,320	680	2,334,192	2,065,713	515	62.47 %	48.13 %	-165
iku-fal17	9,399,840	3,293,922	2,662	256,328	3,383,406	696	97.27 %	-2.72 %	-1,967
mary-spr17	774,553	525,927	62	401,176	371,408	35	48.21 %	29.38 %	-27
muni-fi-spr16	3,114,910	2,705,734	100	1,897,818	1,563,674	58	39.07 %	42.21 %	-42
muni-fsps-spr17	532,588	367,581	34	305,655	249,895	11	42.61 %	32.02 %	-23
muni-pdf-spr16c	16,940,023	3,644,791	4,825	3,222,790	4,043,162	1,984	80.98 %	-10.93 %	-2,841
pu-llr-spr17	4,691,996	1,890,047	289	1,421,551	1,065,820	130	69.70 %	43.61 %	-159
tg-fal17	120,314	126,314	22	36,536	69,179	5	69.63 %	45.23 %	-17
agh-ggos-spr17	13,354,425	2,650,467	1,413	2,339,178	2,212,817	703	82.48 %	16.51 %	-710
agh-h-spr17	10,812,217	1,585,485	5,379	1,429,584	1,571,437	2,524	86.78 %	0.89 %	-2,854
lums-spr18	487,767	435,420	189	114,393	434,291	125	76.55 %	0.26 %	-64
muni-fi-spr17	4,424,788	3,878,005	150	2,622,659	2,187,317	79	40.73 %	43.60 %	-71
muni-fsps-spr17c	13,143,821	862,299	1,497	3,311,773	1,886,161	672	74.80 %	-118.74 %	-825
muni-pdf-spr16	7,815,664	4,602,930	698	3,740,493	3,102,186	450	52.14 %	32.60 %	-248
nbi-spr18	1,220,781	200,008	52	507,099	536,996	28	58.46 %	-168.49 %	-24
pu-d5-spr17	8,142,836	7,684,667	473	3,949,786	3,778,170	145	51.49 %	50.83 %	-328
pu-proj-fal19									
yach-fal17	3,321,465	1,010,854	194	1,166,689	627,610	77	64.87 %	37.91 %	-117
agh-fal17	31,554,446	13,463,124	11,506	9,335,676	9,215,764	4,196	70.41 %	31.55 %	-7,309
bet-spr18	8,344,110	5,193,363	660	3,299,647	2,610,895	512	60.46 %	49.73 %	-148
iku-spr18	6,950,774	2,948,381	2,216	229,650	3,011,718	647	96.70 %	-2.15 %	-1,569
lums-fal17	416,315	432,121	134	102,317	429,509	96	75.42 %	0.60 %	-39
mary-fal18	2,316,193	2,013,267	92	1,315,591	1,212,876	53	43.20 %	39.76 %	-39
muni-fi-fal17	5,407,354	4,950,779	154	3,170,292	2,752,158	93	41.37 %	44.41 %	-61
muni-fspsx-fal17	18,070,645	1,593,509	2,860	5,356,702	2,959,861	1,444	70.36 %	-85.74 %	-1,417
muni-pdfx-fal17	26,529,518	8,310,081	11,008	6,553,907	7,775,828	2,504	75.30 %	6.43 %	-8,504
pu-d9-fal19	75,964,318	23,820,327	5,970	18,137,907	10,765,105	1,978	76.12 %	54.81 %	-3,991
tg-spr18	71,313	71,679	29	59,195	81,334	7	16.99 %	-13.47 %	-22
Average							63.45 %	10.43 %	-1,232

Table 7.8: Size of basic MIP formulation by Holm et al. (2020) compared with the graph-based formulation after presolve. The table includes the time spend on the presolve step of Gurobi.

Time	**1 hour**						**24 hours**					
	Basic MIP			Graph-based MIP			Basic MIP			Graph-based MIP		
Instance	UB	LB	Gap	UB	LB	Gap	UB	LB	Gap	UB	LB	Gap
agh-fis-spr17	-	-	-	-	-	-	-	779	- %	-	**1,028**	- %
agh-ggis-spr17	-	-	-	-	**10,637**	- %	-	**21,703**	- %	-	12,713	- %
bet-fal17	-	-	-	-	-	-	-	-	-	-	**22,248**	- %
iku-fal17	-	-	-	-	-	-	-	10,947	- %	-	**14,930**	- %
mary-spr17	-	2,435	- %	-	**2,116**	- %	15,932	13,212	17.07 %	**15,174**	**14,132**	**6.87 %**
muni-fi-spr16	-	-	-	-	**3,045**	- %	-	3,276	- %	**7,741**	**3,371**	**56.45 %**
muni-fsps-spr17	-	851	- %	**868**	**866**	**0.23 %**	***868***	**868**	**0 %**	***868***	**868**	**0 %**
muni-pdf-spr16c	-	-	-	-	-	-	-	9,196	- %	-	**12,101**	- %
pu-llr-spr17	-	-	-	-	**9,615**	- %	10,710	9,683	9.59 %	**10,107**	**9,856**	**2.48 %**
tg-fal17	***4,215***	**4,215**	**0 %**	***4,215***	**4,215**	**0 %**	***4,215***	**4,215**	**0 %**	***4,215***	**4,215**	**0 %**
agh-ggos-spr17	-	-	-	-	-	-	-	-	-	-	**1,317**	- %
agh-h-spr17	-	-	-	-	-	-	-	5	- %	-	**7,210**	- %
lums-spr18	-	**1**	- %	**201**	**1**	**99.50 %**	**95**	**24**	**74.74 %**	**95**	**24**	**74.74 %**
muni-fi-spr17	-	-	-	-	**1,796**	- %	24,572	2,056	91.63 %	**10,093**	**2,239**	**77.82 %**
muni-fsps-spr17c	-	-	-	-	-	-	-	**923**	- %	**509,503**	815	**99.84 %**
muni-pdf-spr16	-	-	-	-	-	-	-	-	-	-	**11,764**	- %
nbi-spr18	**18,979**	17,438	**8.12 %**	19,848	**17,457**	12.02 %	18,212	17,654	3.06 %	**18,014**	**17,685**	**1.83 %**
pu-d5-spr17	-	-	-	-	-	-	-	4,147	- %	-	**4,529**	- %
pu-proj-fal19												
yach-fal17	-	30	- %	-	**501**	- %	**19,046**	**516**	**97.29 %**	21,736	**516**	97.63 %
agh-fal17	-	-	-	-	-	-	-	-	-	-	**1,125**	- %
bet-spr18	-	-	-	-	-	-	-	-	-	-	**19,326**	- %
iku-spr18	-	-	-	-	-	-	-	14,006	- %	-	**21,707**	- %
lums-fal17	-	196	- %	-	**241**	- %	405	233	42.47 %	**369**	**251**	**31.98 %**
mary-fal18	-	-	-	-	-	-	-	3,009	- %	-	**3,153**	- %
muni-fi-fal17	-	-	-	-	**1,007**	- %	-	1,486	- %	**9,177**	**1,596**	**82.61 %**
muni-fspsx-fal17	-	-	-	-	-	-	-	1,680	- %	-	**1,816**	- %
muni-pdfx-fal17	-	-	-	-	-	-	-	-	-	-	-	-
pu-d9-fal19	-	-	-	-	-	-	-	-	-	-	-	-
tg-spr18	***12,704***	**12,704**	**0 %**	13,576	11,240	17.21 %	***12,704***	**12,704**	**0 %**	***12,704***	**12,704**	**0 %**

Table 7.9: Runtime stats after 1 hour and 24 hours. Best performance is shown in bold. Optimal solutions are in italic. Dash means that the solver is running, but there was no value.

7.10 Future work

The MIP model presented in this paper is the model used for the ITC 2019. This means that after the ITC 2019, some ideas and thoughts have come up. This section will present ideas for future work on the MIP model.
First of all, we have to address the negative reduction in the overall number of variables of the MIP (see Table 7.7). This is due to the number of auxiliary variables generated by the constraints of Section 7.7.1 is larger than auxiliary variables saved by omitting the class pairs in constraints of Section 7.7.3. It should be tested whether it is beneficial to exclude the graphs of Section 7.7.1. If one would like to compare numbers, the numbers are given in Table 7.15 in column *Student Conflicts - Variables.*
The hard `MaxBreaks` constraint can also add edges to the Hard Constraint Class-time Conflict Graph. If the constraint includes only two classes, it is easy to check. If the constraint contains more than two classes, but on any given day in any given week has only two classes possible, a violation of the constraint can be checked, and an edge added accordingly. This also leads to all the so-called *Special Constraints* (`MaxDays`, `MaxDayLoad`, `MaxBreaks`, and `MaxBlock`) are completely covered by the conflict graphs if the constraints contains only two classes, and thus no further constraints are needed.

The soft *Special Constraints* can add edges to the Soft Constraint Class-time Conflict Graph. The logic is the same as for the hard constraints.

The cliques of the clique cover used to generate the clique constraints of Section 7.4.4 are not guaranteed to contain different classes. The clique constraint generated from a clique with only 1 class is weaker than (or best case, as good as) the constraint specifying that all classes must be scheduled. In general, a special clique cover that finds only cliques containing more than one class could be beneficial. Such a special clique cover should only guarantee to cover edges between different classes. The complete bipartite cover is such a special clique cover for a graph containing only two classes.

The hard and soft constraint class-room conflict graphs are completely ignored in modeling the MIP. It should lead to a tighter model if these graphs are used to generate class-room clique and star constraints.

The clique reduction algorithm (Algorithm 6) considers a maximal clique cover generated by a computationally hard heuristic and then searches for cliques containing all vertices of a class. This can be changed to a much faster algorithm. Since all equal-class vertices are pairwise adjacent, considering all vertices of any class will always form a clique. For each class, consider the clique formed by that class's vertices and expand the clique until it is maximal. The vertices used to expand the clique is then removed from the graph. This is a much faster reduction algorithm.

7.11 Concluding remarks

We have presented various methods to reduce the
ITC 2019 instances. The reduction methods showed that redundancy of distribution constraints is present in all instances, but the number of distribution constraints removed varies a lot. The procedure also showed a reduction in the number of decision variables for all instances. In practice, we use the reduction methods to produce a set of reduced input-data files, which is of the same format as the originals. Any algorithm that solves the ITC 2019 problem can use the reduced data files as input, so we have made the reduced data files available on `www.dsumsoftware.com/itc2019`.
Besides reducing the input data, we also proposed a graph-based MIP formulation of the problem. The new MIP formulation combined with the reduced datasets resulted in an at least halving of MIP constraints and a significant reduction of MIP variables compared to the MIP presented by Holm et al., 2020. Even after Gurobi presolve, all the graph-based MIP models proved to be of fewer distribution constraints, and most were of fewer variables. The Gurobi presolve procedure also showed faster completion, which leaves more time for solving when using timelimit as a stopping criterion. The graph-based MIP is thus the state-of-the-art MIP model formulation for the ITC 2019 problem.
We show that solving the graph-based MIP is still hard within 24 hours. We prove three optimal solutions and find solutions for 13 of the 30 instances and lower bounds for 27 of the 30 instances. By finding better upper and lower bounds to more instances, the graph-based MIP shows to outperform the Basic MIP. Our proposed solution method shows well at solving the smaller instances but is not very useful in the majority of the instances. The results show that the model effectively finds and improves the lower bounds, but it is too big for Gurobi to find solutions within the time limit. The many lower bounds combined with the few upper bounds indicate that the MIP might be suitable in a matheuristic.

Appendices

7.A Notations

The notation used is shown in Table 7.10.

Symbol	Description
$\mathcal{D}$	Set of days
$\mathcal{W}$	Set of weeks
$\mathcal{C}$	Set of classes
$\mathcal{C}_\delta$	Set of classes for a distribution constraint δ
$\mathcal{C}_s$	Set of classes a student s can attend
c_i	A specific class with ID $i \in \mathbb{Z}^+$
c_i^{parent}	The parent class of class c_i
c^{limit}	The student limit of class c
$\mathcal{T}$	Set of times
$\mathcal{T}_c$	Set of available times for a class c
t^{length}	The duration in time slots of time t
t^{days}	The set of days $\subseteq \mathcal{D}$ of time t
t^{weeks}	The set of weeks $\subseteq \mathcal{D}$ of time t
T	Set of time slots
$\mathcal{R}$	Set of rooms
$\mathcal{R}_c$	Set of available rooms for a class c
$\mathcal{S}$	Set of students
$\mathcal{S}_c$	Set of students that can attend a class c
Δ	Set of distribution constraints
c_δ	Cost of the distribution constraint δ
G	A graph
$\mathcal{V}$	Set of vertices of a graph G
$\mathcal{E}$	Set of edges of a graph G
ζ_e	The weight on edge e
Ω_G	Set of cliques of a graph G
$\mathcal{S}_G$	Set of stars of a graph G
$\mathcal{O}_G$	Set of odd-cycles of a graph G
$\mathcal{B}_G$	Set of complete bipartite subgraphs of a graph G
$\mathcal{P}$	Set of penalty variables
c_p	Cost of the penalty $p \in \mathcal{P}$
$p_{c,t}$	Penalty of assigning time t to class c
$p_{c,r}$	Penalty of assigning room r to class c
$\mathcal{K}$	Set of courses
Ω_k	Set of configurations of a course k
ψ_t	Objective weight of the time penalties
ψ_r	Objective weight of the room penalties
ψ_δ	Objective weight of the distribution constraint penalties
ψ_s	Objective weight of the student conflict penalties
`D`	Distribution constraint parameter
`G`	Distribution constraint parameter
`M`	Distribution constraint parameter
`R`	Distribution constraint parameter
`S`	Distribution constraint parameter
$x_{c,t,r}$	Binary variable; 1 iff class c is assigned time t and room r
$y_{c,t}$	Binary variable; 1 iff class c is assigned time t
$w_{c,r}$	Binary variable; 1 iff class c is assigned room r
$e_{s,c}$	Binary variable; 1 iff student s is assigned class c
$b_{s,\omega}$	Binary variable; 1 iff student s attends configuration ω
γ_d	Binary variable; 1 iff any class $c \in \mathcal{C}_\delta$ is scheduled on day d
$\phi_{w,d}$	Integer variable; the dayload in week w on day d (for a specific constraint)
$\rho_{w,d,\tau}$	Binary variable; 1 iff a block longer than `M` starts in week w day d on time slot τ (for a specific constraint)
o_{c_i,c_j}	Binary variable; 1 iff classes c_i and c_j overlaps
χ_{s,c_i,c_j}	Binary variable; 1 iff there exists a student conflict between student s and classes c_i and c_j
M	Big-M

Table 7.10: Notations.

7.B Reduced distribution constraints

The number of reduced distribution constraints by each method is shown in Table 7.11.

Instance	Reduced constraints	Single classes	Zero penalty	No violation	Dominated constraints	Dominated special
agh-fis-spr17	44	3	0	36	5	0
agh-ggis-spr17	118	0	0	112	6	0
bet-fal17	92	0	0	83	9	0
iku-fal17	245	1	0	189	47	8
mary-spr17	72	4	0	32	0	36
muni-fi-spr16	74	0	0	50	2	22
muni-fsps-spr17	79	0	0	30	0	49
muni-pdf-spr16c	231	2	0	162	0	67
pu-llr-spr17	118	11	0	41	5	61
tg-fal17	301	2	0	18	0	281
agh-ggos-spr17	75	1	0	60	14	0
agh-h-spr17	27	0	0	27	0	0
lums-spr18	28	0	0	0	2	26
muni-fi-spr17	96	11	0	61	3	21
muni-fsps-spr17c	95	0	0	12	0	83
muni-pdf-spr16	267	0	0	100	0	167
nbi-spr18	8	0	0	7	0	1
pu-d5-spr17	211	0	3	157	17	34
pu-proj-fal19	805	35	10	379	43	338
yach-fal17	58	0	0	25	0	33
agh-fal17	295	4	4	243	43	1
bet-spr18	110	0	0	94	16	0
iku-spr18	308	0	0	255	50	3
lums-fal17	51	2	0	2	1	46
mary-fal18	99	2	0	53	1	43
muni-fi-fal17	87	11	0	47	2	27
muni-fspsx-fal17	182	2	0	82	3	95
muni-pdfx-fal17	628	0	0	364	1	263
pu-d9-fal19	413	9	13	195	61	135
tg-spr18	236	0	0	11	0	225
Total	5,453	100	30	2,927	331	2,065

Table 7.11: Number of reduced distribution constraints.

7.C Reduced variables

The number of vertices reduced by fixed vertices and cliques for each type of graph is given in Table 7.12.

Instance	Total number of reduced vertices	Class-room			Class-time		
		Fixed	Clique	Total	Fixed	Clique	Total
agh-fis-spr17	11,167	8	4	12	9,613	1,542	11,155
agh-ggis-spr17	3,656	0	46	46	2,265	1,345	3,610
bet-fal17	369	0	0	0	319	50	369
iku-fal17	8,616	6	1	7	7,687	922	8,609
mary-spr17	117	0	0	0	93	24	117
muni-fi-spr16	878	0	0	0	678	200	878
muni-fsps-spr17	650	0	0	0	378	272	650
muni-pdf-spr16c	24,247	0	1	1	1,735	22,511	24,246
pu-llr-spr17	38	0	9	9	28	1	29
tg-fal17	4,997	0	0	0	4,792	205	4,997
agh-ggos-spr17	6,458	42	0	42	4,673	1,743	6,416
agh-h-spr17	832	0	38	38	204	590	794
lums-spr18	625	0	0	0	380	245	625
muni-fi-spr17	506	0	0	0	133	373	506
muni-fsps-spr17c	5,131	0	0	0	1,904	3,227	5,131
muni-pdf-spr16	1,185	3	90	93	776	316	1,092
nbi-spr18	721	0	0	0	707	14	721
pu-d5-spr17	746	0	1	1	132	613	745
pu-proj-fal19	3,539	1	56	57	2,140	1,342	3,482
yach-fal17	604	0	0	0	604	0	604
agh-fal17	15,973	95	178	273	12,257	3,443	15,700
bet-spr18	548	0	0	0	350	198	548
iku-spr18	15,891	144	85	229	14,235	1,427	15,662
lums-fal17	340	0	0	0	292	48	340
mary-fal18	258	0	0	0	237	21	258
muni-fi-fal17	297	0	6	6	182	109	291
muni-fspsx-fal17	6,364	0	3	3	1,733	4,628	6,361
muni-pdfx-fal17	30,719	69	37	106	2,447	28,166	30,613
pu-d9-fal19	1,310	57	44	101	852	357	1,209
tg-spr18	5,893	0	0	0	5,400	493	5,893

Table 7.12: Number of reduced vertices in the conflict graphs.

7.D Specified MIP size

This appendix presents the numbers of the specific constraints in the model. Table 7.13 gives the number of constraints generated by the conflict graphs. Table 7.14 shows the number of distribution constraints not modeled by the class-time conflict graph. Table 7.15 presents the number of variables and constraints used to model student sectioning and student conflicts.

Instance	**Cliques**	**Stars**	**Odd-cycles**
agh-fis-spr17	94,122	26,913	1,392
agh-ggis-spr17	30,761	24,617	2,876
bet-fal17	7,394	98,423	4,828
iku-fal17	48,129	16,286	1,709
mary-spr17	3,979	16,507	1,906
muni-fi-spr16	4,032	14,371	1,434
muni-fsps-spr17	6,194	3,995	3,399
muni-pdf-spr16c	35,259	54,230	9,937
pu-llr-spr17	2,615	2,362	636
tg-fal17	5,916	10,876	2,962
agh-ggos-spr17	74,738	56,652	17,019
agh-h-spr17	90,337	41,588	3,108
lums-spr18	23,389	30,640	2,260
muni-fi-spr17	4,066	14,787	1,108
muni-fsps-spr17c	26,378	269	9,886
muni-pdf-spr16	20,464	59,784	13,032
nbi-spr18	18,240	890	5,860
pu-d5-spr17	5,505	72,115	1,884
pu-proj-fal19			
yach-fal17	5,806	6,686	2,058
agh-fal17	262,809	259,079	33,120
bet-spr18	8,703	120,709	6,349
iku-spr18	46,396	9,019	3,076
lums-fal17	25,309	27,592	702
mary-fal18	2,748	13,413	922
muni-fi-fal17	3,652	12,525	1,598
muni-fspsx-fal17	34,690	7,784	14,270
muni-pdfx-fal17	62,007	62,593	28,501
pu-d9-fal19	14,356	64,430	4,738
tg-spr18	5,592	30,846	1,101

Table 7.13: Number of constraints generated from graphs.

Instance	SameRoom	DifferentRoom	SameAttendees*	MaxDays	MaxDayLoad	MaxBreaks	MaxBlock
agh-fis-spr17	52 | 51	0 | 0	547,851 | 74	80 | 48	0 | 0	0 | 89,416	0 | 0
agh-ggis-spr17	1,051 | 126	0 | 0	2 | 0	0 | 24	0 | 0	0 | 5,876	0 | 0
bet-fal17	333 | 510	0 | 431	0 | 0	48 | 0	32 | 0	0 | 0	747,631 | 0
iku-fal17	4,937 | 1,537	0 | 0	1,811 | 6	0 | 0	0 | 0	0 | 0	0 | 0
mary-spr17	157 | 57	0 | 0	337 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-spr16	145 | 9	0 | 0	0 | 0	0 | 0	114 | 0	0 | 0	0 | 14,164
muni-fsps-spr17	116 | 15	0 | 0	7,349 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-pdf-spr16c	132 | 28,555	0 | 0	154 | 0	0 | 0	50 | 1,288	0 | 0	6,992 | 198,389
pu-llr-spr17	424 | 277	0 | 0	8,041 | 0	0 | 0	0 | 0	0 | 6,032	0 | 0
tg-fal17	0 | 0	0 | 0	12 | 2	0 | 0	0 | 0	0 | 0	0 | 0
agh-ggos-spr17	706 | 1	0 | 0	228,985 | 0	16 | 16	0 | 0	8,960 | 0	0 | 0
agh-h-spr17	231 | 27	0 | 0	270,712 | 0	136 | 40	546 | 71	0 | 423,279	0 | 0
lums-spr18	0 | 0	0 | 0	11,652 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-spr17	136 | 11	8 | 0	0 | 0	8 | 0	70 | 0	0 | 0	88,389 | 24,179
muni-fsps-spr17c	0 | 1,489	0 | 0	105,677 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-pdf-spr16	569 | 2,387	0 | 0	785 | 0	0 | 0	286 | 1,541	0 | 0	26,352 | 542,830
nbi-spr18	0 | 0	0 | 0	48,039 | 0	0 | 0	0 | 0	0 | 0	0 | 0
pu-d5-spr17	668 | 1,938	0 | 0	0 | 0	0 | 0	0 | 0	38,220 | 0	0 | 0
pu-proj-fal19	|	|	|	0 | 0	0 | 0	0 | 0	0 | 0
yach-fal17	24 | 1,145	0 | 0	56,939 | 0	0 | 0	2,663 | 0	0 | 0	0 | 0
agh-fal17	3,107 | 2,778	0 | 0	779,835 | 72	328 | 216	975 | 0	165 | 768,791	0 | 0
bet-spr18	249 | 1,417	0 | 1,924	0 | 0	112 | 0	0 | 0	0 | 0	891,280 | 0
iku-spr18	5,444 | 0	0 | 0	5,627 | 0	0 | 0	0 | 0	0 | 0	0 | 0
lums-fal17	0 | 0	0 | 0	595 | 0	0 | 0	0 | 0	0 | 0	0 | 0
mary-fal18	353 | 46	0 | 0	275 | 0	0 | 0	0 | 0	0 | 0	0 | 0
muni-fi-fal17	242 | 0	0 | 0	0 | 0	16 | 0	104 | 53	0 | 0	148,145 | 27,939
muni-fspsx-fal17	68 | 2,235	0 | 0	228,402 | 4	0 | 0	0 | 0	0 | 0	0 | 0
muni-pdfx-fal17	864 | 42,677	0 | 0	354 | 0	0 | 64	422 | 1,614	0 | 0	5,971 | 509,753
pu-d9-fal19	5,127 | 7,695	0 | 0	59,863 | 0	0 | 0	0 | 31	50,022 | 2,484	0 | 0
tg-spr18	1,012 | 3,844	0 | 0	3 | 0	0 | 0	0 | 150	0 | 0	0 | 0

Table 7.14: Number of constraints in the MIP of each type. Data represented Hard | Soft. *SameAttendees reports the number of star constraints used to model time-room overlap.

	Student Sectioning			Student Conflicts	
Instance	$e_{s,c}$	$b_{s,\omega}$	**Constraints**	**Variables**	**Constraints**
agh-fis-spr17	65,376	0	53,277	606,240	745,853
agh-ggis-spr17	116,552	0	57,723	1,037,968	1,130,830
bet-fal17	94,893	15,990	62,440	1,309,441	1,987,349
iku-fal17	-	-	-	-	-
mary-spr17	28,230	0	6,046	164,834	356,253
muni-fi-spr16	64,506	14	7,454	1,501,414	1,910,109
muni-fsps-spr17	20,667	776	7,704	210,147	284,182
muni-pdf-spr16c	80,918	5,246	56,494	1,645,395	2,892,281
pu-llr-spr17	236,980	20,353	103,598	915,435	1,501,613
tg-fal17	-	-	-	-	-
agh-ggos-spr17	56,818	0	45,621	951,361	1,868,758
agh-h-spr17	11,530	0	4,954	201,601	358,802
lums-spr18	-	-	-	-	-
muni-fi-spr17	70,257	60	8,373	2,100,073	2,622,990
muni-fsps-spr17c	17,190	0	10,135	1,631,670	3,293,610
muni-pdf-spr16	103,864	2,822	22,982	2,056,750	3,470,919
nbi-spr18	29,986	0	4,984	386,579	426,993
pu-d5-spr17	442,232	9,886	302,079	3,362,235	3,850,556
pu-proj-fal19					
yach-fal17	28,779	150	25,665	514,828	1,098,858
agh-fal17	305,022	108	198,075	5,265,073	7,085,172
bet-spr18	105,021	14,998	80,461	1,770,579	2,891,833
iku-spr18	-	-	-	-	-
lums-fal17	-	-	-	-	-
mary-fal18	83,731	0	12,970	969,014	1,287,482
muni-fi-fal17	90,997	0	9,991	2,654,329	3,171,684
muni-fspsx-fal17	44,916	194	18,373	2,678,705	5,285,052
muni-pdfx-fal17	241,256	14,776	135,103	3,529,736	5,848,652
pu-d9-fal19	941,391	57,306	513,531	9,537,114	18,012,120
tg-spr18	-	-	-	-	-

Table 7.15: Overview of the number of variables and constraints that are connected to student sectioning and student conflicts.

Acknowledgements

The authors would like to thank Niels-Christian Fink Bagger for providing python code for finding clique covers. Dennis S. Holm's Ph.D. project is part of the Data Science for University Management project (dsumsoftware.com) funded by MaCom A/S and Innovation Fund Denmark (IFD). Rasmus Ø. Mikkelsen's industrial Ph.D. project is funded by IFD. IFD has supported the work solely financially and has not participated in any research-related activities.

References

Bagger, N., Desaulniers, G., and Desrosiers, J. (2019a). "Daily course pattern formulation and valid inequalities for the curriculum-based course timetabling problem". English. In: *Journal of Scheduling* 22.2, pp. 155–172. ISSN: 1094-6136. DOI: 10.1007/s10951-018-0582-0.

Bagger, N., Kristiansen, S., Sørensen, M., and Stidsen, T. (2019b). "Flow Formulations for Curriculum-based Course Timetabling". English. In: *Annals of Operations Research* 280.1-2, pp. 121–150. ISSN: 0254-5330. DOI: 10.1007/s10479-018-3096-4.

Böðvarsdottir, E., Bagger, N., Høffner, L., and Stidsen, T. (2019). *A comprehensive integer programming formulation of the nurse rostering problem in Denmark.* English. Tech. rep.

Bron, C. and Kerbosch, J. (Sept. 1973). "Algorithm 457: Finding All Cliques of an Undirected Graph". In: *Commun. ACM* 16.9, pp. 575–577. ISSN: 0001-0782. DOI: 10.1145/362342.362367.

Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. (2008). "Penalising Patterns in Timetables: Novel Integer Programming Formulations". In: *Operations Research Proceedings 2007.* Ed. by J. Kalcsics and S. Nickel. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 409–414. ISBN: 978-3-540-77903-2.

Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. (2010a). "A supernodal formulation of vertex colouring with applications in course timetabling". In: *Annals of Operations Research* 179(1), pp. 105–130.

Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. (2010b). "Decomposition, reformulation, and diving in university course timetabling". In: *Computers and Operations Research* 37(3), pp. 582–597.

Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. (2012). "A branch-and-cut procedure for the Udine Course Timetabling problem". In: *Annals of Operations Research* 194.1, pp. 71–87. DOI: 10.1007/s10479-010-0828-5.

Gramm, J., Guo, J., Hüffner, F., and Niedermeier, R. (2006). "Data reduction, exact, and heuristic algorithms for clique cover". In: *In Proc. 8th ALENEX.* SIAM, pp. 86–94.

Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2020). *A MIP Formulation of the International Timetabling Competition 2019 Problem.* English. Tech. rep.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Di Gaspero, L., Qu, R., and Burke, E. K. (2010). "Setting the research agenda in automated timetabling: The second international timetabling competition". In: *INFORMS Journal on Computing* 22.1, pp. 120–130. ISSN: 1091-9856. DOI: 10.1287/ijoc.1090.0320.

Müller, T., Rudová, H., and Müllerová, Z. (2018). "University course timetabling and international timetabling competition 2019". In: *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria.* Ed. by E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, pp. 5–31.

Nemhauser, G. and Wolsey, L. (2014). “Strong Valid Inequalities and Facets for Structured Integer Programs”. In: *Integer and Combinatorial Optimization*. John Wiley & Sons, Ltd. Chap. II.2, pp. 259–295. ISBN: 9781118627372. DOI: 10.1002/9781118627372.ch9. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118627372.ch9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118627372.ch9.

Paechter, B., Gambardella, L., and Rossi-Doria, O. (2002). *International timetabling competition 2002*. URL: http://sferics.idsia.ch/Files/ttcomp2002/.

Post, G., Di Gaspero, L., Kingston, J. H., McCollum, B., and Schaerf, A. (2016). “The Third International Timetabling Competition”. In: *Annals of Operations Research* 239.1, pp. 69–75. DOI: 10.1007/s10479-013-1340-5. URL: http://dx.doi.org/10.1007/s10479-013-1340-5.

Chapter 8

A Fix-and-Optimize Matheuristic for the International Timetabling Competition 2019

Rasmus Ørnstrup Mikkelsen[a,b] · Dennis Søren Holm[a] · Matias Sørensen[b] · Thomas Jacob Riis Stidsen[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]MaCom A/S, Vesterbrogade 48, 1., DK-1620 København V, Denmark

Status: Submitted to Journal of Scheduling

Abstract: The International Timetabling Competition 2019 (ITC 2019) poses a novel university timetabling problem composed of assigning classes to times and rooms as well as student sectioning. This paper presents a fix-and-optimize matheuristic tailored to the problem defined by the competition and an initial solution construction heuristic. Both methods used the Mixed Integer Programming (MIP) model defined for the ITC 2019 problem described by Holm et al. (2020a). We detail three implemented neighborhoods and how we dynamically adjust the neighborhood size during the search. The implemented fix-and-optimize matheuristic is evaluated against the MIP on the 30 scoring instances made available in the ITC 2019. The matheuristic has better performance than solving the full MIP alone, finding better solutions on 24 instances. On three instances, fix-and-optimize and the MIP find solutions of equal value, two of which are proved optimal.

Keywords: Mixed Integer Programming · Matheuristic · University Timetabling · International Timetabling Competition 2019 · ITC 2019

8.1 Introduction

University course timetabling is the problem of scheduling a semester of university courses, such that each course event (lectures, lab work, etc.) is assigned a room and time. The timetable should not violate any hard constraints, making it infeasible and unusable. Additionally, the timetable should, as much as possible, observe some soft constraints ensuring timetable quality. Timetables of higher quality have fewer undesirable features and lead to better utilization of resources and more preferable schedules for teachers and students.

The timetabling problem can vary significantly for different universities (Tripathy, 1992). Schaerf (1999) discusses some of the common variants of the university course timetabling problem and solution techniques. The International Timetabling Competition 2019 (ITC 2019) presents a problem description general enough that it can encompass many practical university timetabling problems, as evident by the competition using real-world data from 10 different universities in eight countries on five continents. The problem definition is also novel as it combines student sectioning and classical course event scheduling (Müller et al., 2018). Thus, solutions approaches that work well on this problem definition should be usable in many practical cases.

In this paper, we detail an implemented fix-and-optimize matheuristic for the ITC 2019 problem. The fix-and-optimize matheuristic was presented by Lindahl et al. (2018) to tackle the Curriculum-based Course Timetabling problem presented in the International Timetabling Competition 2007 with excellent results. The matheuristic has been applied to other scheduling problems, such as high-school timetabling (Dorneles et al., 2014) and capacitated lot-sizing (Lang and Shen, 2011; Helber and Sahling, 2010), achieving state-of-the-art results. The method is a type of large neighborhood search using a Mixed Integer Programming (MIP) model of the problem. In each iteration, a subset of decision variables are fixed, and the model is solved, exploring a constrained part of the total solution space. In this work, we use the MIP model for the ITC 2019 described by Holm et al. (2020a).

For the competition, we defined six neighborhoods to used with the fix-and-optimize matheuristic. The neighborhoods are defined by combining three heuristics for choosing classes whose assignments should be allowed to change with two subsets of variables considered for fixing/unfixing. Furthermore, we implemented a heuristic for dynamically updating the neighborhood size throughout the search. The fix-and-optimize implementation shows great performance, outperforming the full MIP on most instances. The algorithm's strength is especially apparent on difficult instances where it finds high-quality solutions, and the MIP finds none.

The paper is organized as follows. Section 8.2 introduces the ITC 2019 problem definition. In Section 8.3, we give a brief overview of the important decision variables of the MIP. Section 8.4 covers the fix-and-optimize matheuristic, including neighborhoods, updating neighborhood size and generating initial solutions. In Section 8.5, we evaluate the implementation through computational results. Section 8.6 concludes.

8.2 Problem definition

Course time and room assignment, and student sectioning are the two main components of the university timetabling problem presented in the ITC 2019. Each course consists of some classes (lectures, exercises, etc.) that must all be assigned one of their defined available times and, if requested, available rooms. The classes of a course are subject to a hierarchical structure, such that they are distributed in configurations and further into subparts. Students request some courses and must be sectioned into classes, such that they attend precisely one class of each subpart of a single configuration of each requested course. A student conflict occurs when a student is assigned two classes that either overlap in time or are placed in rooms sufficiently far apart, so travel time between the classes makes it impossible to reach the second class before it starts.

Another important part of the problem is distribution constraints, which place restrictions on the assignment between two or more classes. There are 19 different types of distribution constraints, and they can either be hard or soft. Distribution constraints are used to, for example, forbid/penalize temporal overlap between specific classes, enforce/prefer some classes use the same room, etc. Most distribution constraint types are evaluated between individual pairs of classes, but four consider all classes included when evaluating the constraint.

A solution to the problem is feasible if it satisfies all hard constraints regarding time and room assignments, hard distribution constraints, and student sectioning. A solution's quality is measured as the weighted sum of soft constraints penalties (each soft constraint is defined with a non-negative integer penalty), the class' time and room assignment (also given a non-negative integer penalty), and the number of student conflicts. Furthermore, categorical weights for time assignment, room assignment, distribution constraints, and student conflicts are defined in each instance, allowing the university to further specify their preferences.

Müller et al. (2018) presents a full problem description.

8.3 Mixed Integer Programming model

The fix-and-optimize matheuristic works by iteratively fixing a subset of decision variables and solving the resulting MIP. The MIP implemented for the ITC 2019 is very extensive, and a full description is beyond the scope of this paper. Here we introduce the decision variables which are considered for fixing. For a description of a basic version of the MIP, we refer to Holm et al. (2020b). Holm et al. (2020a) improve this MIP formulation by using, amongst other things, class-time conflict graphs. We use the MIP formulation by Holm et al. (2020a) in this work and use a class-time conflict graph in one of the matheuristic neighborhoods (see Section 8.4.2).

The problem is defined by a set of classes $\mathcal{C}$ where each class $c \in \mathcal{C}$ must be assigned one of its available times $t \in \mathcal{T}_c$ and rooms $r \in \mathcal{R}_c$. If the class does not need a room then $\mathcal{R}_c = \{\tilde{r}\}$, where $\tilde{r}$ is a dummy-room. Additionally, each student $s \in \mathcal{S}$ must be assigned classes such that they attend their required courses. The problem includes some basic feasibility constraints, e.g., no room double booking, all classes must be scheduled, and student sectioning constraints to observe a specific course/class structure. In addition, distribution constraints restrict/penalize the assignment between two or more classes. We define $\mathcal{C}_\delta$ to be the set of classes that are part of distribution constraint $\delta \in \Delta$, where Δ is the set of all distribution constraints.

Assignment of classes to times and rooms is a major part of the problem, which naturally leads to the main decision variable $x_{c,t,r} \in \{0, 1\}$ defined as

$$x_{c,t,r} = \begin{cases} 1 & \text{if class } c \text{ is scheduled in time } t \text{ in room } r \\ 0 & \text{otherwise} \end{cases}$$

Additionally, we define the auxiliary variable $y_{c,t} \in \{0, 1\}$ as

$$y_{c,t} = \begin{cases} 1 & \text{if class } c \text{ is scheduled in time } t \\ 0 & \text{otherwise} \end{cases}$$

The $y_{c,t}$ variable makes the definition of some constraints much easier and helps reduce the number of non-zeros in the model. We focus on these two sets of variables in the implemented fix-and-optimize matheuristic.

Student sectioning is the other major part of the problem, for which we have defined $e_{s,c} \in \{0, 1\}$ to 1 if student s attends class c and 0 otherwise. However, we observe that generally, the student sectioning part is not difficult compared to the class assignment (except for an extremely large number of students). Therefore we focus on the class assignment, and student sectioning variables are not considered for variable fixing.

8.4 Fix-and-optimize matheuristic

The fix-and-optimize matheuristic can be considered to be a large neighborhood search heuristic. The algorithm takes an initial solution and iteratively improves it by searching a large neighborhood around the current solution. This neighborhood is defined and explored using mixed integer programming by fixing a subset of variables before solving the model using a MIP solver. Thereby the matheuristic is very applicable to problems where small instances can be solved to optimality, but large instances cannot. Fixing variables result in a subproblem with a smaller solution neighborhood, effectively reducing the more difficult MIP to be more easily solved. The fix-and-optimize matheuristic uses this idea by iteratively fixing a subset of variables and solving the resulting MIP. Any improving solution is used as a MIP start in the next iteration.

This approach has a few benefits. Since the algorithm is MIP-based and only fixes/unfixes variables, no direct deliberation is given to the constraints or the structure of the problem. Thereby constraints can be added, removed, or changed, and the algorithm still works. This is in direct contrast to many move-based metaheuristics, in which some moves may rely greatly on specific constraints of the problem and can easily become obsolete as a result of a model change. The fix-and-optimize matheuristic always works on the same model, and therefore it is only necessary to build the MIP model once. Variable fixing is done so that a solution found in one iteration is always feasible regarding the variable fixing in the next iteration. Thereby, the MIP solver is warm started in each iteration, providing great performance benefits. Also, since a MIP is solved in each iteration, we know the subproblem's lower bound, which can be used to guide the search on a higher level. Lastly, an implemented fix-and-optimize will automatically have performance increases as MIP solvers are themselves improved.

As fix-and-optimize can only improve known solutions, it must be given an initial solution. Therefore it is dependent on other methods for generating a starting solution. We discuss the heuristic used for generating starting solutions in Section 8.4.1.

An important aspect of fix-and-optimize is the searched neighborhood, i.e., how we choose variables to fixed. In Section 8.4.2, we discuss our implemented neighborhoods and a strategy for updating the neighborhood size dynamically throughout the search.

The implemented algorithm is described in Algorithm 8. The input is the problem instance, a starting solution, a neighborhood specification, and an initial neighborhood size. First, the algorithm builds the MIP for the given instance and sets the warm start values of the starting solution. In the loop, all variables considered in the given neighborhood, i.e., all candidate variables for fixing, are fixed to their given values. Then the algorithm chooses a subset of variables to unfix, given the neighborhood and neighborhood size. Solving the subproblem updates the best solution if an improving solution is found. Finally, the neighborhood size is updated based on the performance of the optimization process.

Algorithm 8 Fix-and-optimize

```
    Input: Instance $I$, Start solution $S$, Neighborhood $N$ and initial size $P$
    Output: Solution $S$
1: $M \leftarrow$ Build MIP for $I$ and set warm start for $S$
2: while termination condition not met do
3:     Fix all variables considered in $N$
4:     $V$ = GetVariablesToUnfix($N, P$)
5:     Unfix variables $V$ in $M$
6:     $S \leftarrow$ optimize $M$
7:     $P \leftarrow$ UpdateNeighborhoodSize()
8: end while
```

8.4.1 Initial solutions

We generate initial solutions using a simple MIP-based constructive heuristic. We call the heuristic Two-Stage Constructive Algorithm (2SCA) as it first schedules all classes to valid time and room assignments before assigning students to classes. The algorithm uses a modified MIP, where the only objective is minimizing the number of unassigned classes. In the first stage, the algorithm solves the MIP. Once a solution with an objective function value of 0 is found, i.e., all classes are assigned, the algorithm fixes the class assignment variables and adds student sectioning to the problem. In the second stage, the algorithm solves the resulting MIP with a time limit of five minutes and a relative optimality gap of 0.01. If the MIP solver finds no solution within the time limit, it continues until it finds a solution. Algorithm 9 shows the 2SCA.

Algorithm 9 Two-Stage Constructive Algorithm

```
    Input: Instance $I$
    Output: Solution $S$
1:  $M \leftarrow$ Build MIP for $I$ with only unassigned classes objective and no student
    sectioning
2:  Solve $M$ to optimality
3:  Fix class assignment variables and add student sectioning to $M$
4:  $S \leftarrow$ optimize $M$ with time limit of five minutes and relative gap of 0.01,
    or until solution is found
```

8.4.2 Neighborhoods

As stated in Section 8.3, the implemented matheuristic focuses on the classes' time and room assignments, which the developed neighborhoods reflect. Each neighborhood essential defines how to choose classes that should be allowed to be rescheduled. Common for all neighborhoods is that classes are considered course-wise; a list of all courses is randomized, and classes are picked by going through this list and extracting all classes associated with the given course. This continues until the list includes enough classes to satisfy the size parameter (P in Algorithm 8) of the neighborhood. This parameter defines the percent of classes that should be unfixed. Picking all classes associated with a course ensures some measure of relatedness in the unfixed variables.

Each neighborhood separates itself from the others by how it includes extra classes when considering a given course. Three heuristics for picking classes to unfix are defined: Standard (S), Common-Distribution-Constraints (C), and Adjacent-Classes (A). Our neighborhoods use one of these heuristics, and they either consider the class-time assignment variables ($y_{c,t}$) or the class-time-room variables ($x_{c,t,r}$) for fixing/unfixing. Using the $y_{c,t}$ variable allows for room changes of all classes (even those with fixed time) in every iteration, while this is not possible when using the $x_{c,t,r}$ variables. The combination of the class choosing heuristic and assignment variables defines our neighborhoods, resulting in six neighborhoods.

The Standard heuristic goes through the randomized course list, extracting all course classes. No additional classes are chosen.

When extracting course-classes, the Common-Distribution-Constraints heuristic also includes all other classes that share a distribution constraint with any of the course-classes.

The Adjacent-Classes heuristic uses a class-time conflict graph, as used by Holm et al. (2020a), for including additional classes to unfix. When considering a course, all classes with a vertex adjacent to any vertex associated with any of the course-classes are also extracted. Using a conflict graph is similar to considering common distribution constraints but only includes hard constraints and additionally includes information of classes with fixed times. Thereby, this heuristic focuses on unfixing class assignment variables that interdependently affect feasibility.

8.4.2.1 Updating neighborhood size

It is vital to continually control the neighborhood's size as this has a significant effect on the difficulty of the subproblems. The goal is to have problems that are neither too hard nor too easy to solve; the neighborhood should be large enough to include new (hopefully improving) solutions, but not so large that they are too time-consuming to find. Hence we dynamically update the neighborhood size throughout the search to strike such a balance. Dynamically updating the neighbor size is especially important for problems such as the ITC 2019, as the MIP complexity varies greatly from instance to instance and thereby requires vastly different parameter settings.

We focus the implementation on solving the ITC 2019 problem in the competitive setting defined for the release of the 10 *Late* instances, which is 10 days before the competition deadline. Therefore, we have tuned the implementation towards a 10 days time frame to find the best possible solutions for each instance. Some instances lead to MIP models that are very difficult to solve, resulting in iterations where the solver spends much time on presolving the model and solving the root node linear program (LP) relaxation. Therefore, we implement a conservative neighborhood size updating scheme, such that we are more willing to decrease the neighborhood size than increase it. Additionally, if the solution process has not completed the presolve and solved the root node LP after an hour, the neighborhood size is drastically reduced. The solving process is given a time limit of one hour to reach the branch and bound tree's root node and 30 minutes to explore the branch and bound tree.

Algorithm 10 details how the neighborhood size is updated based on the solver performance. We update the neighborhood size by considering how quickly the solver progresses through the solving process, if any improving solutions are found, and the optimality gap. The decision conditions are checked in a prioritized manner, such that when any condition is satisfied, the corresponding decision is made immediately. We adjust the neighborhood size in steps of Δ^P (set to 5%), and the initial size is $P = 25\%$. The update algorithm has a boolean "ShouldIncrease" memory (H), which we add to in each iteration; true if the size should increase and false otherwise. The neighborhood size is updated conservatively, favoring decreasing by doing so immediately upon such a decision. Alternatively, if the neighborhood size should increase, we check H and increase the size if three of the last five memory inputs are true, i.e., if the last few iterations heavily favor increasing.
We also consider a special case where the solving process does not reach the root node of the branch and bound tree within an hour. In such cases, the model is far too difficult to solve, and we drastically reduce the neighborhood

by dividing the current size by two and rounding down to the nearest divisor of Δ^P. Additionally, we reset the H since the information from previous iterations is of little value following such a substantial change. Any time we change P, we ensure that it stays within its natural limits of 0.05 and 1, corresponding to 5% (the value of Δ^P) and 100%.

Algorithm 10 UpdateNeighborhoodSize

```
    Given: Neighborhood size $P$ and boolean "ShouldIncrease" memory $H$
 1: if Not reached root node in one hour then
 2:     $P \leftarrow P/2$ rounded down to nearest divisor of $\Delta^P$ without remainder
 3:     Reset $H$
 4: else if Time to root node $>$ 30 minutes then
 5:     Decrease($P$, $H$)
 6: else if Found no improving solutions then
 7:     if Gap $< 0.05$ then
 8:         Increase($P$, $H$)
 9:     else
10:         Decrease($P$, $H$)
11:     end if
12: else
13:     Add false to $H$
14: end if
15: return $P$
```

Algorithm 11a Decrease(P, H)

```
1: $P \leftarrow P - \Delta^P$
2: Add false to $H$
```

Algorithm 11b Increase(P, H)

```
1: Add true to $H$
2: if at least 3 of last 5 inputs in $H$ are true then
3:     $P \leftarrow P + \Delta^P$
4: end if
```

8.5 Computational results

In this section, we evaluate the algorithm and compare each defined neighborhood's performance through computational tests on all 30 ITC 2019 instances of the Early, Middle, and Late categories. All computational tests are performed in a cluster setting using 64bit computers running Scientific Linux 7.7. The machines are equipped with 256GB RAM and two Intel Xeon E5-2650 v4 CPUs clocked at 2.20GHz. We use Gurobi 9.0 as the MIP solver.

To generate starting solutions for the fix-and-optimize algorithm, we run the 2SCA on each instance. Table 8.1 shows the found solution's objective value and time required by the algorithm. For 20 out of the 30 instances, the 2SCA finds a starting solution in less than an hour. For the remaining instances, 2SCA finds a solution within 6 hours, except for *bet-fal17*, which requires approximately 25.5 hours. The time-consuming and challenging part of the constructive heuristic is the first step of finding a feasible schedule. For *bet-fal17*, a solution for the second step is found in roughly 10 minutes.

Instance	Objective	Time (s)
agh-fis-spr17	41,247	8,414.5
agh-ggis-spr17	180,008	330.3
bet-fal17	378,894	92,165.3
iku-fal17	163,853	7,056.2
mary-spr17	78,394	150.4
muni-fi-spr16	17,490	129.2
muni-fsps-spr17	122,153	74.9
muni-pdf-spr16c	538,628	3,458.7
pu-llr-spr17	94,177	323.0
tg-fal17	25,754	23.4
agh-ggos-spr17	86,140	2,745.2
agh-h-spr17	61,941	13,321.5
lums-spr18	2,043	385.8
muni-fi-spr17	15,612	185.5
muni-fsps-spr17c	557,983	916.2
muni-pdf-spr16	309,812	1,482.8
nbi-spr18	130,637	90.7
pu-d5-spr17	44,794	461.3
pu-proj-fal19	574,825	12,337.2
yach-fal17	18,489	1,154.3
agh-fal17	524,880	16,248.2
bet-spr18	441,251	21,501.7
iku-spr18	193,034	6,570.5
lums-fal17	2,698	341.0
mary-fal18	35,635	771.6
muni-fi-fal17	17,734	199.9
muni-fspsx-fal17	933,433	2,081.2
muni-pdfx-fal17	838,957	9,184.4
pu-d9-fal19	326,980	4,654.7
tg-spr18	98,946	17.7

Table 8.1: An overview of the solutions produced by the 2SCA.

Using different heuristics for choosing classes to unfix is only relevant if these heuristics achieve different results. We compare the chosen classes to unfix from each heuristic pair, using the same course list as input and extracting

Instance	S-C	S-A	C-A
agh-fis-spr17	40.6	45.0	72.6
agh-ggis-spr17	45.3	65.3	52.0
bet-fal17	50.5	50.9	86.8
iku-fal17	34.5	33.1	69.0
mary-spr17	48.5	59.5	63.7
muni-fi-spr16	46.1	57.3	62.1
muni-fsps-spr17	45.4	45.8	79.4
muni-pdf-spr16c	37.3	42.2	73.6
pu-llr-spr17	62.4	56.5	54.9
tg-fal17	39.4	27.2	81.1
agh-ggos-spr17	44.2	44.3	83.2
agh-h-spr17	40.1	43.0	85.8
lums-spr18	35.8	37.3	94.0
muni-fi-spr17	59.5	60.5	78.2
muni-fsps-spr17c	57.6	58.7	90.9
muni-pdf-spr16	41.9	42.7	80.2
nbi-spr18	50.7	50.5	85.1
pu-d5-spr17	81.8	84.9	78.9
pu-proj-fal19	68.5	57.9	51.8
yach-fal17	59.6	67.2	71.4
agh-fal17	41.3	54.1	68.1
bet-spr18	47.1	48.3	89.0
iku-spr18	33.7	32.3	61.8
lums-fal17	37.3	38.0	99.1
mary-fal18	55.4	69.9	62.0
muni-fi-fal17	55.1	58.3	69.7
muni-fspsx-fal17	33.7	47.8	44.4
muni-pdfx-fal17	32.9	39.1	58.6
pu-d9-fal19	71.8	72.8	69.6
tg-spr18	27.0	30.8	50.8
Average	47.5	50.7	72.3

Table 8.2: The percentage overlap of classes for each neighborhood pair using the same course list input and extracting 25% of all classes.

25% of all instance classes. Table 8.2 shows the average percentage overlap in chosen classes for five different course list inputs. We expect some overlap since we give the same input courses to each heuristic, and all heuristics, when considering a course, includes all its classes. The overlap between the Standard heuristic and the others is approximately 50%. As mentioned in Section 8.4.2, the Adjacent-Classes heuristic is partly similar to the Common-Distribution-Constraints heuristic, as it focuses mainly on hard distribution constraints, which leads to an increased overlap, averaging for all instances 72.3% overlap.

Three instances have more than 90% overlap, explained by the instances having a large percentage of hard constraints used in the conflict graph. For example, in *lums-fal17*, 91% of its distribution constraints are hard and of the types SameTime, SameDays, NotOverlap, and SameAttendess; all contributing to the conflict graph.

To compare neighborhood and algorithm performance, we run the fix-and-optimize algorithm using each neighborhood on all instances. We use the initial solutions found by the 2SCA to start the fix-and-optimize algorithms. When solving subproblems in the fix-and-optimize algorithm, the MIP focus parameter is set to "feasibility" (`MIPFocus`=1 for Gurobi), prioritizing finding feasible solutions quickly. We compare the fix-and optimize algorithm to the full MIP formulated by Holm et al. (2020a), solved using default settings. To have a fair comparison, we warm-start the MIP with solutions from the 2SCA. All runs are allowed to run for 24 hours using 4 cores.

Table 8.3 shows the best solutions obtained by a single run of each method within 24 hours. We show the best solution found for each instance in bold. Using the lower bound provided by the MIP, we know that we have solved two instances to optimality: *tg-fal17* and *muni-fsps-spr17*. No data could be collected for the *pu-proj-fal19* instance because this instance results in a MIP that requires more RAM than the 256GB available. The *pu-proj-fal19* instance is by far the biggest, including more courses, classes, rooms, students, and distribution constraints than any other instance. We disregard *pu-proj-fal19* for the remainder of this section since data could not be collected.

TThe table shows that the neighborhoods based on the Standard heuristic have better performance than the others, finding a majority of the best solutions. Only on two instances did the Common-Distribution-Constraint heuristic-based neighborhoods find the best solutions alone, and the Adjacent-based neighborhoods found four. The success of the Standard heuristic compared to the others may be attributed to the fact that all methods are given a starting solution of low quality. Perhaps the Standard heuristic is very good for an initial dive, while the others are better for thoroughly searching the solution space, once improving solutions are more difficult to find. Neighborhoods that use the $x_{c,t,r}$ variables for fixing seem to have slightly better performance than those fixing $y_{c,t}$, especially for larger and more difficult instances. The success on difficult instances when focusing on the $x_{c,t,r}$ variables makes sense, as the resulting subproblems are more constrained and more easily solved.

The full MIP found the best solutions for 6 instances during the 24 hour time limit. However, the MIP did not find any solutions for 10 instances, contrasted by the 6 fix-and-optimize runs, which all found solutions to all instances.

The MIP and all fix-and-optimize neighborhoods, except for C_Y, produced some best results in the given 24 hour time limit. Although C_Y generally performs worse, it found the second-best solution on instances *yach-fal17* and *iku-spr18*. It is hard to predict a priori which methods work well on a given instance. Since C_Y is competitive on at least a few instances, we assume the fix-and-optimize matheuristic (using all neighborhoods) and the full MIP are suitable methods for tackling the ITC 2019 problem. Figure 8.1 shows the solution values per time for each method on four different instances. The figures

	Fix-and-optimize						MIP	
Instance	S_Y	S_X	C_Y	C_X	A_Y	A_X	UB	LB
agh-fis-spr17	4,465	**4,313**	5,365	4,863	7,106	4,342	-	1,027.4
agh-ggis-spr17	43,422	41,442	41,151	**39,992**	44,996	46,450	171,732	12,148.0
bet-fal17	317,932	**307,407**	330,946	322,632	327,399	330,977	-	22,248.0
iku-fal17	33,614	25,563	27,102	25,082	23,713	29,695	**22,391**	14,982.3
mary-spr17	**14,978**	15,393	19,711	20,379	16,119	16,773	15,076	14,072.8
muni-fi-spr16	4,024	4,022	4,260	4,304	**3,901**	4,056	7,109	3,372.0
muni-fsps-spr17	**868**	**868**	2,725	879	**868**	882	**868**	868.0
muni-pdf-spr16c	113,896	**83,800**	136,520	100,889	130,958	97,782	-	12,100.2
pu-llr-spr17	10,283	10,190	31,384	47,363	10,199	15,433	**10,062**	9,918.1
tg-fal17	**4,215**	**4,215**	**4,215**	**4,215**	**4,215**	**4,215**	**4,215**	4,215.0
agh-ggos-spr17	8,873	**5,751**	11,588	7,277	11,061	7,194	58,241	1,318.9
agh-h-spr17	**27,973**	29,760	32,266	31,693	29,058	31,582	-	7,221.8
lums-spr18	96	100	181	212	100	121	**95**	24.0
muni-fi-spr17	4,210	4,163	5,221	5,425	**3,957**	4,403	14,125	2,226.0
muni-fsps-spr17c	24,487	**7,452**	28,831	12,075	23,741	15,575	358,752	1,031.2
muni-pdf-spr16	50,228	**29,193**	67,750	44,495	52,280	36,395	290,574	10,320.3
nbi-spr18	18,018	**18,016**	18,631	18,332	18,125	18,158	18,053	17,640.6
pu-d5-spr17	21,196	22,333	22,463	25,200	**20,550**	23,567	26,382	4,542.7
pu-proj-fal19	-	-	-	-	-	-	-	-
yach-fal17	**2,727**	3,623	2,732	3,608	3,429	4,648	18,410	516.0
agh-fal17	433,349	**296,364**	391,456	309,942	372,608	321,452	-	-
bet-spr18	**380,854**	384,705	398,680	398,731	404,982	395,770	-	19,326.0
iku-spr18	44,836	33,580	33,568	**31,854**	34,922	40,057	42,260	20,995.5
lums-fal17	372	**358**	428	469	373	367	384	250.2
mary-fal18	**5,398**	6,437	10,363	9,050	5,465	7,225	16,919	3,142.0
muni-fi-fal17	3,660	**3,513**	3,773	4,387	3,642	3,653	-	1,583.6
muni-fspsx-fal17	120,139	**40,531**	219,073	77,283	156,616	156,573	-	1,815.4
muni-pdfx-fal17	400,517	**165,225**	448,051	199,875	526,180	216,559	-	-
pu-d9-fal19	137,941	131,452	177,993	225,346	135,060	**125,555**	-	-
tg-spr18	**12,704**	**12,704**	**12,704**	**12,704**	**12,704**	17,454	**12,704**	12,501.9

Table 8.3: Best solution and bound values found in 24 hours. Bold results show the best for that instance.

are ordered in increasing MIP sizes, with approximately 400.000 constraints and variables, 1.9 million constraints and 1.6 million variables, 4.3 million constraints and 4.0 million variables, and 10.0 million constraints and variables, respectively.

This figure shows that the solving the MIP is competitive on instances resulting in smaller MIPs. On the *mary-spr17* instance, the MIP follows a very similar solution value progression as the fix-and-optimize matheuristics. Furthermore, the bound provided by the MIP provides valuable information about the state of the search, so we know we are close to an optimal solution after 24 hours. For the other instances, the MIP is less competitive. For *muni-fi-spr16* the MIP still provides bound information and finds solutions, but the matheuristics clearly outperform these solutions. For *pu-d5-spr17* the MIP finds a single solution of comparable quality after approximately 13 hours and some bound information. Finally, for *agh-fal17*, the MIP cannot find any solutions or even provide bound information within 24 hours.

Figure 8.1 also shows how the fix-and-optimize runs using neighborhoods that consider $x_{c,t,r}$ variables find more and better solutions on the larger in-

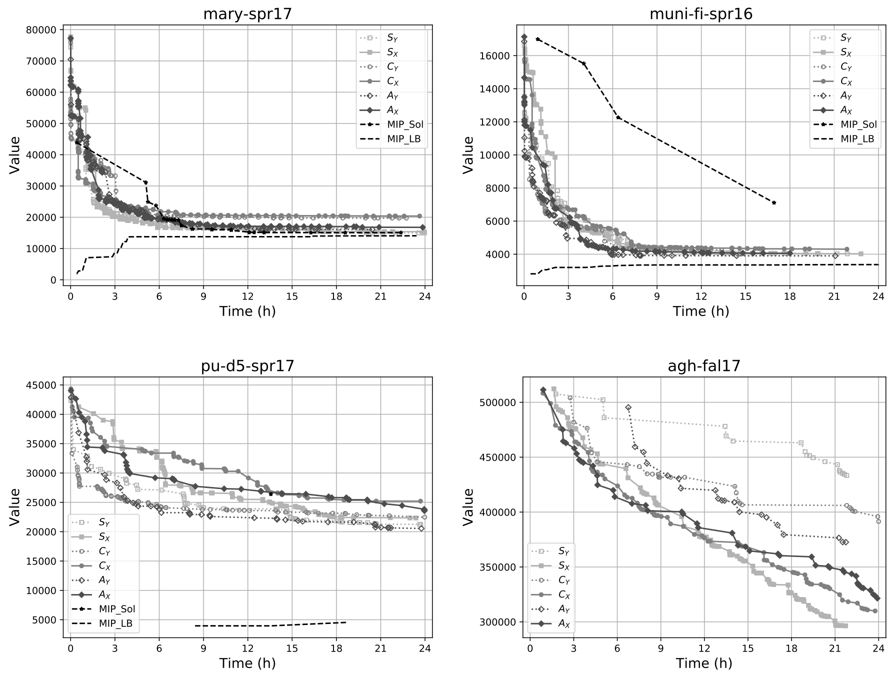


Figure 8.1: Solution values per time for each solution method on *mary-spr17*, *muni-fi-spr16*, *pu-d5-spr17* and *agh-fal17*.

stance *agh-fal17*. Additionally, the figure shows how on the *pu-d5-spr17* instance, neighborhoods using $y_{c,t}$ outperforms their counterparts using $x_{c,t,r}$. While this instance results in a MIP that is smaller than the *agh-fal17* instance, it is not a trivial instance; it is one of the instances with the most classes and students. Further arguing for the complexity of *pu-d5-spr17* is that the MIP only finds one solution within 24 hours. However, it is not only MIP size that solely dictates which neighborhood type yields better performance with the fix-and-optimize matheuristic. For example, the *muni-pdf-spr16c* and *muni-pdf-spr16* instances results in MIPs of similar size as *pu-d5-spr17*, but for both instances neighborhoods using $x_{c,t,r}$ are consistently better.

The fix-and-optimize implementation starts with a neighborhood size of 25% and is changed dynamically throughout the search. Figure 8.2 shows the solution values and neighborhood size for the fix-and-optimize runs using the S_Y neighborhood on *mary-spr17* and *pu-d5-spr17*. On *mary-spr17*, the neighborhood size constantly remains 25% for the first 30 iterations, as the matheuristic has success finding improving solutions. From iteration 27 and on, many subproblems have an initial low gap, and no solutions are found. Therefore the neighborhood size is expanded, increasing the likelihood that the neighborhood contains an improving solution.

Conversely, the heuristic iteratively decreases neighborhood size for *pu-d5-spr17* until it has the minimum possible size of 5%. All decisions to decrease

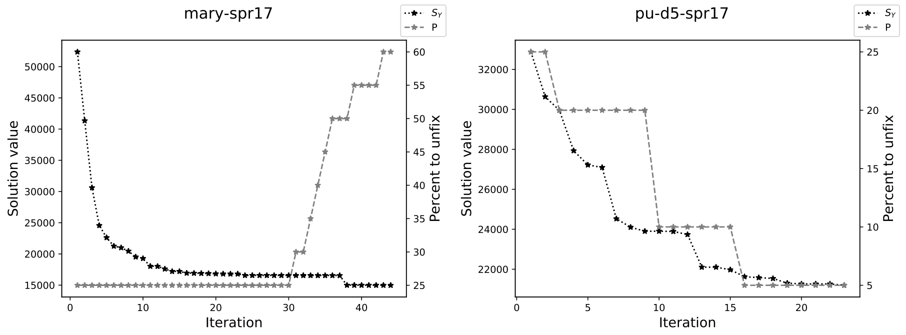


Figure 8.2: Solution values and neighborhood size (P) per iteration for the fix-and-optimize using the S_Y neighborhood on *mary-spr17* and *pu-d5-spr17*.

the neighborhood size are due to the MIP solver taking a long time to reach the branch and bound tree. In iteration 10, the neighborhood size is halved because the solver did not reach the branch and bound tree within an hour. However, in iteration 20 and 22, the neighborhood size updating heuristic determines to increase the size. These two examples show a general trend for neighborhood size progression. For many instances, the neighborhood size initially updates to a value fitting early in the search where many improving solutions exist. For those instances that reach a point where the search stagnates, the heuristic increases the neighborhood size.

8.6 Conclusion

We have proposed a fix-and-optimize matheuristic and solution constructive heuristic specialized for the ITC 2019 problem. Six different neighborhoods are defined and tested on all regular competition instances, and most show promising results, with no neighborhood being clearly better than the others. Computational results also show that the implemented fix-and-optimize algorithm is better at finding feasible solutions of high quality than solving the full MIP using a commercial solver alone. Only on a few of the smaller instances does the MIP have better or comparable results, and on 10 instances, the MIP was not able to find any solutions.

In the future, the implementation can be improved by refining how the neighborhood size is dynamically updated. For example, it could be beneficial to update the neighborhood size more fluidly than the implemented fixed increase/decrease of 5%. The algorithm should be allowed to make smaller/bigger changes in neighborhood size. Furthermore, the algorithm should try to set an appropriate initial neighborhood size based on the instance instead of using a fixed initial size of 25%. Especially for very hard instances, this leads to wasted time, as the algorithm has to spend several iterations decreasing the neighborhood size to an appropriate level.

Additionally, it could be a good idea to dynamically update other algorithm parameters during the solution process, especially the subproblem's time limit. For example, in cases where the algorithm converges and it becomes more difficult to find improving solutions, the algorithm should be given more time to explore each subproblem thoroughly. In such cases, the neighborhood size updating heuristic could also use other criteria for making decisions. For example, when the search converges towards the optimal solution, an initial low gap is unavoidable and should not be used in the decision-making process.

Acknowledgements

Dennis S. Holm's Ph.D. project is part of the Data Science for University Management project (dsumsoftware.com) funded by MaCom A/S and Innovation Fund Denmark (IFD). Rasmus Ø. Mikkelsen's industrial Ph.D. project is funded by IFD. IFD has supported the work solely financially and has not participated in any research-related activities.

References

Dorneles, Á. P., Araújo, O. C. de, and Buriol, L. S. (2014). "A fix-and-optimize heuristic for the high school timetabling problem". In: *Computers & Operations Research* 52, pp. 29–38.

Helber, S. and Sahling, F. (2010). "A fix-and-optimize approach for the multi-level capacitated lot sizing problem". In: *International Journal of Production Economics* 123.2, pp. 247–256.

Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2020a). "A Graph Based MIP Formulation of the International Timetabling Competition 2019". Manuscript submitted for publication.

Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2020b). *A MIP Formulation of the International Timetabling Competition 2019 Problem.* English. Tech. rep.

Lang, J. C. and Shen, Z.-J. M. (2011). "Fix-and-optimize heuristics for capacitated lot-sizing with sequence-dependent setups and substitutions". In: *European Journal of Operational Research* 214.3, pp. 595–605.

Lindahl, M., Sørensen, M., and Stidsen, T. R. (2018). "A fix-and-optimize matheuristic for university timetabling". In: *Journal of Heuristics* 24.4, pp. 645–665.

Müller, T., Rudová, H., and Müllerová, Z. (2018). "University course timetabling and international timetabling competition 2019". In: *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria.* Ed. by E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, pp. 5–31.

Schaerf, A. (1999). "A survey of automated timetabling". In: *Artificial intelligence review* 13.2, pp. 87–127.

Tripathy, A. (1992). "Computerised decision aid for timetabling—a case analysis". In: *Discrete applied mathematics* 35.3, pp. 313–323.

Chapter 9

A Parallelized Matheuristic for the International Timetabling Competition 2019

Rasmus Ørnstrup Mikkelsen[a,b] · Dennis Søren Holm[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]MaCom A/S, Vesterbrogade 48, 1., DK-1620 København V, Denmark

Status: Submitted to Journal of Scheduling

Abstract: The International Timetabling Competition 2019 (ITC 2019) presents a novel and generalized university timetabling problem composed of traditional class time and room assignment as well as student sectioning. In this paper, we present a parallelized matheuristic tailored to the ITC 2019 problem. The matheuristic is composed of multiple methods using the graph-based Mixed Integer Programming (MIP) model defined for the ITC 2019 problem by Holm et al. (2020). We detail all methods included in the parallelized matheuristic and how they collaborate. The parallelized matheuristic includes two methods for producing initial solutions and uses the fix-and-optimize matheuristic presented by Mikkelsen et al. (2020) to find improving solutions. Additionally, the full MIP is used to calculate lower bounds. We detail how the methods collaborate through solution sharing and a diversification scheme for when the search stagnates. Furthermore, we detail how the parallelized matheuristic decomposes the problem for instances with a substantial number of students. Components of the parallelized matheuristic are evaluated using the 30 scoring instances made available in the ITC 2019. The complete parallelized matheuristic performs well, even solving some instances to proven optimality, and we see that solution sharing especially has a significant effect on the search. The quality of the parallelized matheuristic is further exemplified by the fact that it is the winning algorithm of the competition.

Keywords: Mixed Integer Programming · Parallelized Matheuristic · Fix-and-optimize · University Timetabling · International Timetabling Competition 2019 · ITC 2019

9.1 Introduction

University timetabling is a complex scheduling problem that all universities must regularly solve in practice. The classical university course timetabling problem consists of developing a semester timetable such that all course events (lectures, exercises, etc.) are assigned a room and time. The goal is to make a feasible high-quality timetable. A timetable is feasible if it satisfies all hard constraints, and its quality is measured by the violations of the soft constraints.

The timetabling problem definition differs greatly between universities due to traditions and political structure, making it difficult to make a single competitive solution approach (Tripathy, 1992). University timetabling has received a lot of research attention, see e.g. Lewis (2008), Burke and Petrovic (2002), Schaerf (1999), and Carter and Laporte (1998), partly due to the organization of international timetabling competitions. The International Timetabling Competition of 2007 presented a formal description of the Curriculum-based Course Timetabling (CB-CTT) problem together with 21 benchmark instances from the University of Udine (Di Gaspero et al., 2007). Such a competition allows researchers to compare results on the same problem and instances and is exceedingly valuable for improving the state-of-the-art.

This paper describes a parallelized matheuristic for the university timetabling problem presented in the International Timetabling Competition 2019 (ITC 2019). The ITC 2019 presents a novel university timetabling problem definition that contains both course assignments and student sectioning. The competition is evaluated using 30 very different real-world data instances collected from 10 institutions around the world (Müller et al., 2018b).

Our solution approach combines multiple methods based on Mixed Integer Programming (MIP) in a parallelized setting. Most notably, we use simultaneous searches using the fix-and-optimize matheuristic of Mikkelsen et al. (2020) combined with solving the full MIP using a black-box solver. We use the matheuristics to find high-quality solutions and primarily use the MIP to provide lower bound information. Solutions are shared between methods to accelerate the combined search, and a diversification scheme is implemented to escape local optima. Additionally, the parallelized matheuristic includes a special setup for data instances with a substantial number of students.

The proposed solution approach presents a general framework that can be applied to any MIP. One simply needs to define neighborhoods for the fix-and-optimize matheuristics and methods for constructing initial solutions.

However, since many MIPs need to be solved, the performance of the framework is directly tied to the strength of the MIP formulation.

The paper is organized as follows. Section 9.2 presents the ITC 2019 problem. Section 9.3 gives a short introduction to the MIP. Section 9.4 briefly covers the fix-and-optimize matheuristic. Section 9.5 describes the complete parallelized matheuristic. Section 9.6 evaluates the method through computational results and Section 9.7 concludes.

9.2 Problem definition

The ITC 2019 problem defines a novel university timetabling problem combining classical course time and room assignment and student sectioning. Courses consist of one or more classes that must all be assigned one of their predefined times and available rooms (if requested), each defined with a non-negative integer penalty. The course classes are separated into configurations, which are further divided into subparts, and this hierarchical structure is important for student sectioning. Each student requests some courses to which the student must be assigned. A valid student-course assignment observes that the student attends exactly one class of each subpart of a single configuration and that class attendance limits are not exceeded. The configurations and subparts are designed so that students enrolled in a course are guaranteed to attend a valid combination of the course's classes.

Distribution constraints place restrictions on the assignment between two or more classes and can either be hard or soft. There are 19 different types of distribution constraints, such as forbid/penalize temporal overlap between classes, enforce/prefer classes to be scheduled on the same day, etc. Most distribution constraint types are evaluated pairwise for the affected classes, but four are evaluated using all classes for which they are defined. Soft distribution constraints are defined with a non-negative penalty value.

A feasible solution satisfies all hard constraints concerning class, time, and room assignment, student sectioning, and all hard distribution constraints. The quality of a solution is measured by the weighted sum of soft distribution constraint penalties, class time and room assignment, and the number of student conflicts. A student conflict occurs when a student is assigned to two classes that overlap in time or are scheduled such that it is impossible to reach the second class in time due to travel distance. The university defines categorical weights for time assignment, room assignment, student sectioning, and distribution constraints according to their priorities and preferences.

The competition consists of 30 instances (see Table 9.1) made publicly available in three separate stages: Early, Middle, and Late. The competition's goal is to find the best possible solutions for each instance before the competition deadline. The ranking of competitors is determined using a scoring scheme where instances released later in the competition are given a higher score. The Late instances are released 10 days before the final deadline.

The competition instances vary greatly in terms of size, characteristics, and complexity. For example, one of the Early instances, *tg-fal17*, consists of only 36 courses with 711 classes, 15 rooms, and no student sectioning. This is contrasted by the Middle instance *pu-proj-fal19*, which includes a staggering 2,839 courses with 8,813 classes, 768 rooms, and 38,437 students. However, the complexity of an instance is not only determined by the number of classes, rooms, students, etc. Both the types and number of defined distribution constraints have a significant effect.

The problem presented by the ITC 2019 is fascinating as it provides a unified problem definition that can encompass the practical timetabling problem arising at many different universities. This enables the competition to use real-world data from 10 different universities in eight countries on five continents. The data has been collected using the timetabling system UniTime (Müller et al., 2018a). The ITC 2019 problem is based on the model used in UniTime, with some simplifications to reduce modeling complexity while retaining hardness. For a full problem description, we refer the reader to Müller et al. (2018b).

9.3 Mixed Integer Programming model

The central part of our solution approach is the graph-based MIP (referred to as the *full MIP*) defined by Holm et al. (2020), which includes two main binary decision variables; $x_{c,t,r}$ for assigning classes to times and rooms, and $e_{s,c}$ for assigning students to classes. The full MIP uses the auxiliary variable $y_{c,t}$, which is set by $x_{c,t,r}$ and represents class-time assignments, to simplify formulations of constraints. A complete description of the full MIP is very comprehensive and beyond the scope of this paper. For a description of the full MIP, we refer the reader to Holm et al. (2020).

For most instances, the student sectioning part is not difficult compared to the class assignment. However, for instances like *pu-proj-fal19*, which has 38,437 students, it becomes quite complex and computer memory intensive. Therefore, we define a version of the MIP that applies the student sectioning to a known, feasible timetable. When the timetable is fixed, we know which class pairs overlap in time or have a large enough travel distance such that a student conflict is possible. Thereby, we can avoid generating a lot of redundant student sectioning variables and constraints. Additionally, we can ignore the class assignment part of the problem, including distribution constraints, resulting in a MIP that is much more manageable. We refer to this MIP as *the student sectioning MIP*. The student sectioning MIP is thus similar to the full MIP, but where the full MIP has variables related to the timetable, these are fixed and thereby represented as parameters in the student sectioning MIP. Thereby, the student sectioning MIP has only one main decision variable, the student-class assignment variable $e_{s,c}$.

9.4 Fix-and-optimize matheuristic

One of the main methods we use to find high-quality solutions is the fix-and-optimize matheuristic, which is a large neighborhood search heuristic. The neighborhood is defined and explored using mixed integer programming by fixing a subset of variables and solving the model using a MIP solver. Fixing

Instance	Courses	Classes	Rooms	Students
agh-fis-spr17	340	1,239	80	1,641
agh-ggis-spr17	272	1,852	44	2,116
bet-fal17	353	983	62	3,018
iku-fal17	1,206	2,641	214	-
mary-spr17	544	882	90	3,666
muni-fi-spr16	228	575	35	1,543
muni-fsps-spr17	226	561	44	865
muni-pdf-spr16c	1,089	2,526	70	2,938
pu-llr-spr17	687	1,001	75	27,018
tg-fal17	36	711	15	-
agh-ggos-spr17	406	1,144	84	2,254
agh-h-spr17	234	460	39	1,988
lums-spr18	313	487	73	-
muni-fi-spr17	186	516	35	1,469
muni-fsps-spr17c	116	650	29	395
muni-pdf-spr16	881	1,515	83	3,443
nbi-spr18	404	782	67	2,293
pu-d5-spr17	212	1,061	84	13,497
pu-proj-fal19	2,839	8,813	770	38,437
yach-fal17	91	417	33	821
agh-fal17	1,363	5,081	327	6,925
bet-spr18	357	1,083	63	2,921
iku-spr18	1,290	2,782	208	-
lums-fal17	328	502	97	-
mary-fal18	540	951	93	5,051
muni-fi-fal17	188	535	36	1,685
muni-fspsx-fal17	515	1,623	33	1,152
muni-pdfx-fal17	1,635	3,717	86	5,651
pu-d9-fal19	1,154	2,798	224	35,213
tg-spr18	44	676	24	-

Table 9.1: Some characteristics of all 30 competition instances.

variables results in a subproblem that is more easily solved. Adaptively updating the number of variables to fix, i.e., the neighborhood size, the matheuristic can keep subproblems manageable, making it useful for both smaller (and more easily solved) instances, as well as larger and more difficult instances. A short introduction to the implemented fix-and-optimize matheuristic is given here, and a comprehensive description is given by Mikkelsen et al. (2020).

In the implemented fix-and-optimize matheuristic, variables $x_{c,t,r}$ and $y_{c,t}$ are used for fixing. When fixing $y_{c,t}$ variables, all class assignments are allowed room changes (including those fixed in time) in every iteration, while this is not possible when using $x_{c,t,r}$ for fixing. Thereby, fixing $x_{c,t,r}$ as opposed to $y_{c,t}$ results in more constrained subproblems.

Three different heuristics are used to choose which classes should be allowed to be rescheduled (the associated assignment variables remain unfixed in the subproblem). Common for each heuristic is that they consider courses in random order and extract all classes of each course until enough classes have been chosen. The Standard (S) heuristic simply extracts course-classes and nothing else. The Common-Distribution-Constraint (C) heuristic additionally includes all classes that share a distribution constraint with any of the course-classes. The final heuristic, Adjacent-Classes (A), uses a class-time conflict graph, as presented by Holm et al. (2020), which provides information about what pairwise class-time assignments are impossible. Thus, when considering a course, all classes with a vertex adjacent to any vertex associated with any of the course-classes are also extracted.

The combination of class choosing heuristic and variable set used for fixing defines our neighborhoods results in six neighborhoods in total.

9.5 Parallelized matheuristic

In this section, we describe our parallelized matheuristic, where multiple search methods are run in parallel. Section 9.5.1 details the two heuristics used to find initial solutions, which are especially important for the fix-and-optimize matheuristic. Section 9.5.2 describes how solutions are shared between the search methods. Section 9.5.3 covers the implemented diversification scheme, which is invoked when the search stagnates. Section 9.5.4 describes some additions that are focused on handling instances with a substantial number of students. Section 9.5.5 provides an overview of the complete parallelized matheuristic setup.

9.5.1 Initial solutions

To generate initial solutions, we use two simple MIP-based constructive heuristics. One is the Two-Stage Constructive Algorithm (2SCA) introduced by Mikkelsen et al. (2020), and the other is a new Three-Stage Constructive Algorithm (3SCA). In these methods, we utilize that there always exists a feasible student sectioning since student conflicts are defined as soft violations. For valid data, the hard constraints associated with student sectioning, i.e., class limits and valid student-course-class assignments, can always be satisfied. However, it may result in a great number of student conflicts. Thereby, given any feasible timetable, students can always be added afterwards without losing feasibility.

The 2SCA (shown in Algorithm 12) constructs a feasible solution in two stages by first generating a feasible timetable and then adding student sectioning. To generate a feasible timetable, the 2SCA uses a modified MIP where students and soft constraints are ignored, and classes are allowed to be unscheduled. The only objective of the MIP is then the number of unscheduled classes. Thus a solution with an objective value of 0 is a feasible timetable for the full MIP. When such a solution has been found, the student sectioning MIP is used to create a student sectioning for the found timetable. The result is a feasible timetable with student sectioning.

Algorithm 12 Two-Stage Constructive Algorithm

```
    Input: Instance $I$
    Output: Solution $S$
 1: $M_1 \leftarrow$ Build MIP for $I$ with only unassigned classes objective and no
    student sectioning
 2: $S_{M_1} \leftarrow$ Solve $M_1$ to optimality
 3: $M_s \leftarrow$ Build student sectioning MIP for $I$ from $S_{M_1}$
 4: $S \leftarrow$ optimize $M_s$ with time limit of five minutes and relative gap of 0.01,
    or until solution is found
```

Algorithm 13 Three-Stage Constructive Algorithm

```
    Input: Instance $I$
    Output: Solution $S$
 1: $M_1 \leftarrow$ Build MIP for $I$ with no assignment of rooms, no soft distribution
    constraints and no student sectioning
 2: $M_2 \leftarrow$ Build MIP for $I$ with no soft distribution constraints and no student
    sectioning
 3: $S_{M_1} \leftarrow$ Solve $M_1$
 4: Fix time-assignment of $M_2$ from $S_{M_1}$
 5: $S_{M_2} \leftarrow$ Solve $M_2$
 6: while $M_2$ is infeasible do
 7:     Cut off $S_{M_1}$ from $M_1$
 8:     $S_{M_1} \leftarrow$ Solve $M_1$
 9:     Fix time-assignment of $M_2$ from $S_{M_1}$
10:     $S_{M_2} \leftarrow$ Solve $M_2$
11: end while
12: $M_s \leftarrow$ Build student sectioning MIP for $I$ from $S_{M_2}$
13: $S \leftarrow$ Optimize $M_s$ with time limit of five minutes and relative gap of 0.01,
    or until solution is found
```

The 3SCA (shown in Algorithm 13) constructs a feasible solution in three stages. The first stage assigns classes to times by using a modified MIP that ignores both soft constraints and students and allows classes not to be assigned a room. Stage two fixes the class-time-assignment of stage one and forbids classes with unassigned rooms (unless the class should not be assigned a room according to the instance data). Solving the MIP of stage two results either in a feasible timetable or in an infeasible model. In the case of an infeasible model, we go back to stage one and cut off the class-time-assignment and resolve. When a feasible timetable has been found, we proceed to stage three, where we use the student sectioning MIP to add student sectioning to the found timetable.

When solving the first MIP in 2SCA, the MIP solver is allowed to run until a solution is found with an objective value of 0 (solved to optimality). When solving the models in stage one and two of 3SCA, the MIP solver is given a time limit of five minutes and a relative optimality gap of 0.05. However, if no solution is found within the time limit, the solving process is continued until a solution is found. In both 2SCA and 3SCA, students are added to the found timetables by solving the student sectioning MIP with a time limit of five minutes and a relative optimality gap of 0.01, or until a solution is found.

The 3SCA is introduced because it considers quality to some extent. While 2SCA randomly chooses a feasible timetable, 3SCA uses a greedy approach for assigning times and then rooms. This method might prove to be slower, but it should have a higher probability of providing initial solutions of better quality compared to 2SCA.

The 2SCA and 3SCA are not bound to be run only once and can be set to run until certain other stopping criteria (e.g., time limit, number of solutions) are met. In such cases, they add a cut after a feasible timetable has been found, forbidding the previous solution. This forces the algorithm to find a new feasible timetable. The cut used in the 2SCA is shown below. S_n is the set of $x_{c,t,r}$ set to 1 in stage one in the n'th iteration of 2SCA. The cut for 3SCA uses $y_{c,t}$ instead of $x_{c,t,r}$.

$$\sum_{S_n} x_{c,t,r} \leq |S_n| - 1$$

Furthermore, the 2SCA and 3SCA can skip the last stage and simply generate timetables without student sectioning. As mentioned, the timetables are feasible in the full MIP since the student sectioning cannot make the complete solution infeasible (can only add penalty through student conflicts). This is an important feature which is used in section 9.5.4.

9.5.2 Solution sharing

For the fix-and-optimize matheuristic, we use several neighborhoods that have a varied performance on different instances (Mikkelsen et al., 2020). We have no clear way to determine a priori which neighborhood is better suited for a given instance and therefore opt to use them all, with a single fix-and-optimize process using each neighborhood. Naturally, it would be beneficial to share good solutions between each search, to ensure that none falls significantly behind. This also enables us to use each neighborhood to continually improve the best-found solution with their different focuses and strengths.

In such a scheme, it is important to strike a balance between allowing each fix-and-optimize process to search its own neighborhood and simultaneously not waste time where no improvements are possible. Therefore, each fix-and-optimize search checks if there is a better known solution every five iterations. If such a solution exists, it is used going forward. Each fix-and-optimize iteration uses a maximum of 1.5 hours, and such a check is therefore done at most every 7.5 hours.

The full MIP is also run in parallel to the fix-and-optimize searches. As the MIP is only competitive with fix-and-optimize on smaller instances (Mikkelsen et al., 2020), we run the MIP with a focus on improving the lower bound, as this information is very valuable for evaluating the state of the search. The MIP continually watches for and sets any new best known solution, as this can be useful for managing the size of the branch and bound tree and speeds up the search. However, the MIP solver requires some time for reading and setting new solutions, which does take time away from improving the bound.

9.5.3 Diversification

Even with solution sharing, there is a risk of the fix-and-optimize searches getting stuck in a local optimum. Therefore we include a diversification scheme to attempt to escape from such a local optimum and continue the search.

The focus of the parallelized matheuristic is on the final 10 days of the competition, where the goal is to find the best possible solutions on the Late instances. With this in mind, we have chosen that the fix-and-optimize processes enter "diversification" mode after 12 hours with no improving solution. When diversifying, the fix-and-optimize processes "scramble". They each reset to a random new initial solution (generated by the 2SCA or 3SCA) and use that as a starting point for a new search. Solution sharing is turned off, and each fix-and-optimize process uses all three heuristics for choosing classes to unfix, chosen at random in each iteration. All heuristics are used to counter the effects of turning solution sharing off by not limiting a search using a heuristic that may not be well suited for a given instance. No change is made to the variable set used for fixing.

Algorithm 14 AbandonDiversifySolution(S, S^*, B, F)

Input: New solution S, best known solution S^*, list of previous solution values B, boolean F stating if failed to get new starting solution

Output: Boolean value stating whether to get a new solution for diversification

```
1: $B^* \leftarrow \min(B)$                          ▷ Best solution found in this search
2: Add $S$ to $B$
3: if $F$ and $S \geq B^*$ then
4:     return true
5: end if
6: if $S < B^*$ then
7:     $F \leftarrow$ false
8:     return false
9: end if
10: if length of $B \leq 5$ then
11:     return false
12: end if
13: $L \leftarrow$GetConsecutiveNonImprovementLimit($B^*, S^*$)
14: return If last $L$ solutions in $B$ have equal values
```

Each fix-and-optimize process needs to determine when to abandon the current diversification search and reset to a new initial solution. Once more, it is important to strike a balance between giving the search a chance and not wasting too much time where success is unlikely. Algorithm 14 details the heuristic used for determining when to abandon the current diversification search and try again from another initial solution. If no new starting solution is available when

Algorithm 15
GetConsecutiveNonImprovementLimit(B^*, S^*)

```
    Input: Current best solution $B^*$, best known solution $S^*$
    Output: Integer number of consecutive non-improvement limit
 1: $G \leftarrow$ gap from $B^*$ to $S^*$
 2: if $G \geq 20\%$ then
 3:     return 2
 4: else if $G \geq 10\%$ then
 5:     return 3
 6: else if $G \geq 5\%$ then
 7:     return 4
 8: else
 9:     return 5
10: end if
```

the algorithm determines to restart, the search simply continues and resets to a new solution when one becomes available; unless an improvement has been found in the meantime. This is the role of input parameter F, which is used in line 3 and updated in line 7. Lines 3-5 checks if it was previously decided to abandon the search ($F = \textit{true}$), and the current solution is non-improving. Lines 6-9 come into play when the new solution improves, and F is updated to allow the search to continue normally and returns that the search should not be abandoned. The diversifying search is always allowed to continue for at least 5 iterations, which is ensured by lines 10-12. Finally, a limit is imposed on the number of consecutive iterations where no improving solutions are found. This limit is dependent on the gap between the best solution of the current diversifying search and the best known solution. Algorithm 15 shows the used limits (ranging from 2 to 5), which are defined to allow for more leniency as the search gets closer to the best known solution value. Whenever the search is reset to a new starting solution, all parameters are reset to their default values, and the diversification-solution value memory list (B) are cleared.

When a new best known solution is found, all fix-and-optimize processes jump to this solution and reset all parameters. The search then continues normally with solution sharing turned on, and each fix-and-optimize process uses its designated heuristic for choosing classes to unfix.

9.5.4 Large Student Sectioning

Some competition instances include an excessive number of students, which causes problems regarding computational memory and time when using the full MIP. This is at least the case with *pu-proj-fal19*, which has the largest number of courses, classes, and students of all competition instances. Holm et al. (2020) noted that the full MIP for *pu-proj-fal19* requires more than 256GBs of RAM. Therefore, we include some additional methods and a special setup for such instances, where we allow for decoupling timetable development and student sectioning.

By separating the class assignment and student sectioning problems, the respective MIPs become much more manageable. However, solving the class assignment problem without considering students comes at the cost of no look-ahead, which may result in poor solutions. It may be the case that high-quality timetables disregarding students yield many student conflicts, which is especially unfortunate if student conflicts have a relatively large weight. Therefore, the aim of the extension discussed here is not to pursue the best possible solutions but to improve the chances of finding feasible solutions for large and difficult instances.

For instances with 30,000 students or more, we invoke a special setup with a few changes/additions. We denote this setup Large Student Sectioning (LSS). One addition is the Add Student Sectioning to Timetables (ASST) method, which takes a timetable as input and adds students using the student sectioning MIP described in Section 9.3. Valid timetables without students are generated using the 2SCA and 3SCA heuristics by each skipping their last stage. Furthermore, we run two additional fix-and-optimize processes, which both ignore student sectioning. One uses $y_{c,t}$, and the other uses $x_{c,t,r}$ for variable fixing. These two searches are included to produce timetables of high quality (disregarding student sectioning). In order to produce a lot of feasible timetables, these two fix-and-optimize processes are always in "diversification" mode, meaning that there is no solution sharing, and they use all heuristics for getting classes to unfix. Additionally, they are more willing to abandon a diversification starting solution, doing so immediately when a non-improving solution is found after the first five iterations.

The ASST method prioritizes using timetables produced by the two special fix-and-optimize processes. If none are available, it chooses a timetable produced by 2SCA or 3SCA. The student sectioning MIP is solved with a time limit of 10 minutes but continues until a solution is found. We run five ASST in parallel, and they share a list of timetables that have had students added, such that the same timetable is only considered once. We run multiple ASST processes because we have observed cases where the ASST method has difficulty keeping up with the influx of input solutions.

In summary, for instances with 30,000 students or more, we run the default setup with the addition of LSS. This means that the constructive heuristics skip their last stage of adding students, which is instead handled by an additional five ASST processes. In an attempt to improve the relatively random timetables produced by 2SCA and 3SCA, we include two fix-and-optimize searches that ignore student sectioning. Thereby, we run the default setup with some additional methods to increase the likelihood of finding feasible solutions.

9.5.5 Final setup

Figure 9.1 shows the general flow of the combined parallelized matheuristic. Rounded boxes and rectangles represent methods/algorithms and types of solutions, respectively. Given a new instance, the instance is first reduced as described by Holm et al. (2020), to remove redundancies and tighten the instance data. Afterward, all algorithms are started in parallel: 2SCA and 3SCA begin finding initial solutions, a MIP solver starts solving the full MIP, and all fix-and-optimize processes prepare for an initial solution by building the MIP. The fix-and-optimize processes look for the best known solution, typically an initial solution produced by 2SCA or 3SCA, but could be found using the full MIP. Both 2SCA and 3SCA continue producing multiple initial solutions. The search continues for as long as time allows or until the full MIP has proven optimality.

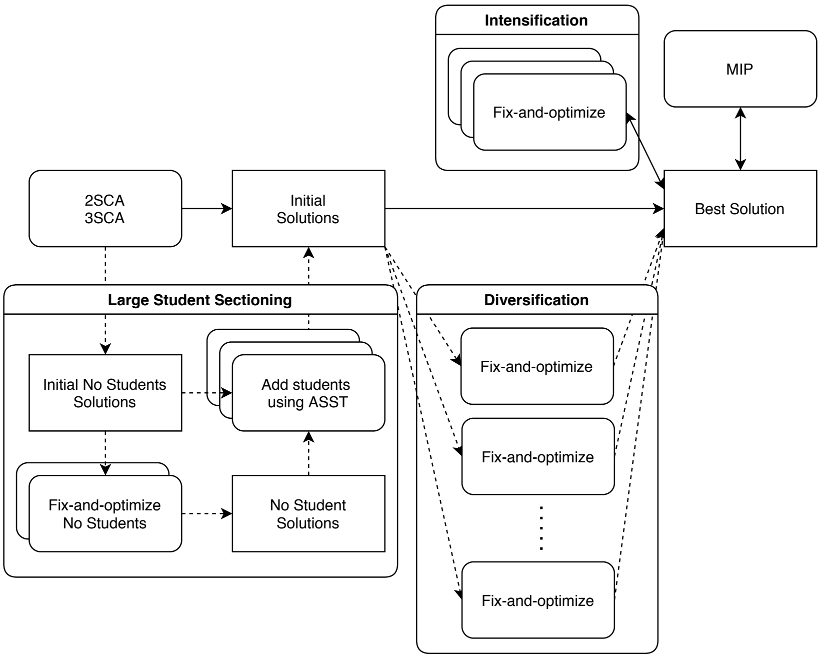


Figure 9.1: Flowchart illustrating the parallelized matheuristic, showing the solution flow, method collaboration and the changes imposed by the “Large Student Sectioning” and “Diversification” modes.

While the search is moving forward, the fix-and-optimize processes collectively remain in a state of intensification by checking and potentially resetting to the best known solution every five iterations. If the best known solution has not been updated in 12 hours, then the fix-and-optimize matheuristics begin diversification by abandoning solution sharing, and each process separately starts a new search using a new initial solution. When the best known solution is improved (by any method), the fix-and-optimize processes reset to this solution and resume solution sharing (intensification).

If the given instance has 30,000 or more students, the 2SCA and 3SCA skip adding students to the feasible timetables they find. Instead, they produce initial solutions without students, which are used as input for two fix-and-optimize processes that also ignore students and the five ASST processes that solve the student sectioning problem using the student sectioning MIP. Once students have been added to these timetables, the solutions enter the solution flow as initial solutions.

In the competition, we used Gurobi 8.1.1 as the MIP solver and had the following setup:

- One 2SCA and one 3SCA both producing at most 200 initial solutions and both using 4 threads
- One MIP solve with focus on bound using 16 threads
- Six fix-and-optimize processes, one using each class choosing heuristic (Standard, Common-Distribution-Constraint, Adjacent) and variable set ($x_{c,t,r}$, $y_{c,t}$) combination, all using 4 threads

For instances with 30,000 or more students, we had the following changes/additions:

- The solution limit for 2SCA and 3SCA increased to 1,000 initial solutions (without students)
- Two fix-and-optimize processes ignoring students both using 4 threads
- Five ASST processes adding students to valid timetables, each using a single thread

9.6 Computational results

This section evaluates the parallelized matheuristic through computational tests on all 30 instances used for ranking in the ITC 2019. All computational tests are performed in a cluster setting on 64bit computers running Scientific Linux 7.7 equipped with 256GB RAM and two Intel Xeon E5-2650 v4 CPUs clocked at 2.20GHz. We use Gurobi 9.0 as the MIP solver using four threads unless stated otherwise.

9.6.1 Initial solutions

We use both 2SCA and 3SCA to generate initial solutions. Table 9.2 shows the objective cost and time to find the first solution for each of the two methods on all instances. As expected, the 2SCA finds solutions quicker than 3SCA, but they are generally of lesser quality. 3SCA is only quicker to find the first solution on eight instances, but in these cases, 2SCA requires at most 449 seconds more. Conversely, 2SCA is many hours faster than 3SCA on some instances. Instances *iku-spr18* and *muni-pdfx-fal17* represent extreme cases, where 3SCA is not able to find a single feasible solution within 10 days, and 2SCA does so in 6,571 and 9,184 seconds respectively. For most other instances, 3SCA produced a solution of higher quality. The notable exception is the three *muni-fsps* instances, where the 2SCA produced better solutions. For these three instances much greater weight is given to student conflicts than all other timetable penalties. Thus, the extra time spent in 3SCA on improving the timetable's quality without considering students is counterproductive as it results in an increase in student conflicts, which is much more costly.

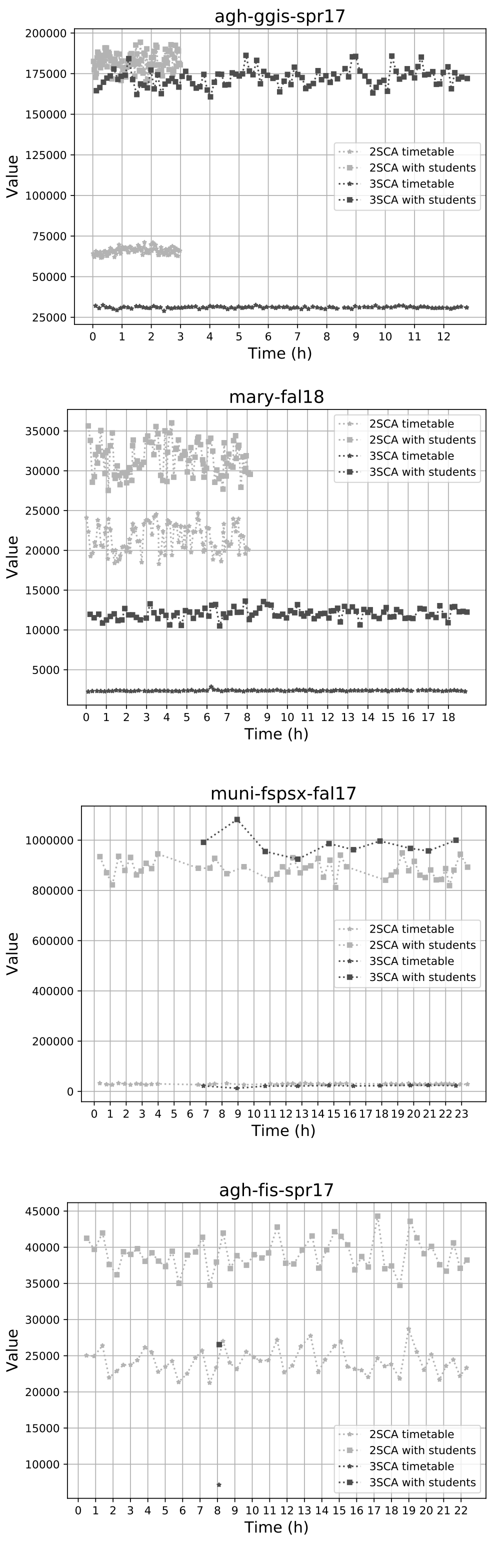


Figure 9.2: Solution values per time for solutions generated by 2SCA and 3SCA on *agh-ggis-spr17*, *mary-fal18*, *muni-fspsx-fal17* and *agh-fis-spr17*.

	2SCA		3SCA	
Instance	Objective	Time (s)	Objective	Time (s)
agh-fis-spr17	41,247	**8,415**	**26,542**	35,295
agh-ggis-spr17	180,008	**330**	**169,344**	452
bet-fal17	378,894	**92,165**	**330,667**	231,477
iku-fal17	163,853	**7,056**	**148,662**	137,215
mary-spr17	78,394	**150**	**36,732**	222
muni-fi-spr16	17,490	129	**9,208**	**126**
muni-fsps-spr17	**122,153**	**75**	163,127	154
muni-pdf-spr16c	538,628	**3,459**	**349,360**	59,581
pu-llr-spr17	94,177	323	**23,059**	**215**
tg-fal17	25,754	**23**	**8,830**	25
agh-ggos-spr17	86,140	**2,745**	**58,346**	4,464
agh-h-spr17	61,941	**13,322**	**51,981**	25,302
lums-spr18	2,043	**386**	**467**	1,022
muni-fi-spr17	15,612	186	**9,949**	**122**
muni-fsps-spr17c	**557,983**	**916**	650,626	1,357
muni-pdf-spr16	309,812	**1,483**	**125,864**	3,358
nbi-spr18	130,637	**91**	**42,334**	106
pu-d5-spr17	44,794	461	**28,216**	**236**
pu-proj-fal19	578,484	**12,718**	**172,711**	167,162
yach-fal17	18,489	1,154	**17,714**	**676**
agh-fal17	524,880	**16,248**	**454,145**	58,081
bet-spr18	441,251	**21,502**	**386,775**	96,631
iku-spr18	**193,034**	**6,571**	-	-
lums-fal17	**2,698**	**341**	2,759	2,061
mary-fal18	35,635	772	**12,673**	**705**
muni-fi-fal17	17,734	200	**13,162**	**108**
muni-fspsx-fal17	**933,433**	**2,081**	991,450	25,346
muni-pdfx-fal17	**838,957**	**9,184**	-	-
pu-d9-fal19	326,980	4,655	**107,983**	**4,205**
tg-spr18	98,946	**18**	**74,126**	23

Table 9.2: An objective and time comparison of the first solution found by 2SCA and 3SCA on all 30 instances. Bold results are the best objective/time for that instance.

Figure 9.2 shows the solution values with and without students for 2SCA and 3SCA for instances *agh-ggis-spr17*, *mary-fal18*, *muni-fspsx-fal17*, and *agh-fis-spr17*. Both 2SCA and 3SCA are run using a time limit of 24 hours and a solution limit of 100 solutions. All plots show that 2SCA is faster at generating solutions but varies more in the objective value. This is expected since the timetable produced in the first stage of 2SCA is found without considering any normal timetable related objectives. Since 3SCA does consider the timetable quality, the solutions it produces are of more similar quality, which is especially seen in the plots for *agh-ggis-spr17* and *mary-fal18*. These two plots also show a great difference in solution quality between the two algorithms, especially on *mary-fal18*, where 3SCA finds much better solutions. The plot for *muni-fspsx-fal17* shows the special case where 2SCA consistently finds better solutions. Although 3SCA, for this instance, finds timetables (not including students) with an objective value between 5,000 and 10,000 less than 2SCA, the cost of adding students to these timetables outweighs these differences. The plot for *agh-fis-spr17* shows how 3SCA is only able to find a single solution within the 24 hour time limit, as opposed to 2SCA, which finds 52. However, this single solution is of much great quality than any produced by 2SCA.

Although 2SCA and 3SCA have a varying performance for different instances, the general observation is that 2SCA is quicker to find solutions but of lesser quality compared to 3SCA. The aim of these two methods is to quickly find a starting solution that the fix-and-optimize matheuristic can improve. For this purpose, 2SCA is generally the better method to use. However, 3SCA is not without merit, as the secondary purpose of the generated initial solutions is to function as starting solutions for fix-and-optimize during diversification. In this case, it is typically better to use a starting solution of higher quality.

9.6.2 Solution sharing

In the parallelized matheuristic, we share solutions between the fix-and-optimize processes and the full MIP to have a majority of computational time spent where there is a chance of moving the search forward. This is especially true for fix-and-optimize, where an individual search may fall behind because its given neighborhood is a poor fit for that instance or simply because of misfortune. By periodically moving the search to the best known solution, the chance that it can contribute to the overall search is increased. While we also continually feed the best known solution to the full MIP, this is mostly to help prune nodes in the branch-and-bound tree.

Table 9.3 shows the results of running six fix-and-optimize matheuristics (one for each neighborhood) and the full MIP on all instances except *pu-proj-fal19* with and without solution sharing for 24 hours. The 2SCA solution from Table 9.2 is used as the initial solution. For this test, we ignore *pu-proj-fal19* as the resulting MIP is too memory-intensive for our test computers with 256GB RAM. Sharing solutions yields better results on all instances, except for *muni-fsps-spr17*, *tg-fal17* and *tg-spr18* which are solved to optimality in both cases. When solutions are shared, the best solution value is improved by an average of 10.42%. There is no major difference in the best bounds obtained, which is not surprising considering the short run time. Solution sharing is beneficial for improving the bound when new solutions can be used for pruning nodes in the branch-and-bound. The MIP only begins branching for nine of the instances. All others are working in the root node of the tree; some even still solving the root node linear relaxation. For three instances, solution sharing resulted in a worse lower bound. This is a consequence of the MIP solver having to spend time parsing and handling passed solutions and changing the search strategy as a result of the available solutions. For *pu-llr-pr17* and *tg-spr18* the MIP solver managed to respectively explore 529 and 10,706 additional nodes when not sharing solutions. With solution sharing turned on, a total of 46 and 7 solutions were passed to the MIP solver for the two instances, respectively. Both with and without solution sharing *iku-spr18* the MIP solver was still exploring the root node after 24 hours. With solution sharing, a total of 10 solutions were passed and the MIP solver spent time improving those solutions, forsaking time from improving the lower bound.

By sharing solutions, the search's overall speed is improved, consistently yielding better results within the first 24 hours. And although no improvement is shown in the lower bound, this would be expected for longer run times.

9.6.3 Large Student Sectioning

The MIP for *pu-proj-fal19* is massive, requiring a significant amount of RAM, and making it intractable to solve both the full MIP as well as the greatly constrained MIP used in fix-and-optimize. To counter this problem, we introduced some additions that decouple timetable development and student sectioning. By using these methods, we are able to find feasible solutions for *pu-proj-fal19*. The additions are invoked for instances with 30,000 students or more, meaning that *pu-proj-fal19* and *pu-d9-fal19* are the only affected instances.

Table 9.4 shows the best solution values found in 24 hours using three different setups on *pu-proj-fal19* and *pu-d9-fal19* five times. The tested setups are as follows. The default "parallelized matheuristic (PM)" where we run 2SCA, 3SCA, six fix-and-optimize, and one full MIP. "Large Student Sectioning (LSS)" where we have 2SCA and 3SCA skipping student sectioning, two fix-and-optimize improving those solutions (ignoring student sectioning), and five ASST adding students to the produced timetables. The last setup is "PM + LSS" in which we combine "PM" and "LSS."

The best solutions produced for *pu-proj-fal19* using the LSS setup vary greatly in solution value; the average value is 209,636, but the greatest difference is 51,781. The solutions found for *pu-d9-fal19* using LSS are much more consistent (the largest difference is 7,344) but of much worse quality than those produced by the other setups. For *pu-d9-fal19*, there is no discernible advantage to using the LSS methods combined with the default setup. Ultimately, both "PM" and "PM + LSS" reach similar value solutions within the first 24 hours of the search, with an average solution value of 50,179 and 49,619, respectively, an absolute difference of only 560. The solutions found in these tests for *pu-d9-fal19* are much better than the one shown in Table 9.3 since these tests also use 3SCA for generating initial solutions, which for *pu-d9-fal19* produces much better solutions than 2SCA.

9.6.4 Full setup with diversification

Here we run the full setup on a few select instances for a total of seven days. The setup closely mimics what was used in the competition: 2SCA and 3SCA producing up to 200 initial solutions, a single MIP solve (using 4 threads instead

	No sharing		Sharing		Improvement (%)	
Instance	UB	LB	UB	LB	UB	LB
agh-fis-spr17	4,313	1,028	3,958	1,028	8.2	0.0
agh-ggis-spr17	39,992	12,148	38,106	12,291	4.7	1.2
*bet-fal17**	307,407	22,248	307,019	22,248	0.1	0.0
iku-fal17	22,391	14,983	19,344	15,049	13.6	0.4
mary-spr17	14,978	14,084	14,930	14,123	0.3	0.3
muni-fi-spr16	3,901	3,372	3,854	3,374	1.2	0.1
muni-fsps-spr17	868	868	868	868	0.0	0.0
muni-pdf-spr16c	83,800	12,101	62,243	12,101	25.7	0.0
pu-llr-spr17	10,062	9,919	10,042	9,868	0.2	-0.5
tg-fal17	4,215	4,215	4,215	4,215	0.0	0.0
agh-ggos-spr17	5,751	1,319	4,115	1,319	28.4	0.0
agh-h-spr17	27,973	7,222	26,044	7,222	6.9	0.0
lums-spr18	95	24	95	24	0.0	0.0
muni-fi-spr17	3,957	2,226	3,852	2,256	2.7	1.3
muni-fsps-spr17c	7,452	1,032	5,312	1,073	28.7	3.8
muni-pdf-spr16	29,193	10,321	23,525	10,321	19.4	0.0
nbi-spr18	18,016	17,641	18,014	17,646	0.0	0.0
pu-d5-spr17	20,550	4,543	18,000	4,543	12.4	0.0
pu-proj-fal19	-	-	-	-	-	-
yach-fal17	2,727	516	1,523	516	44.2	0.0
*agh-fal17**	296,364	1,125	270,125	1,125	8.9	0.0
*bet-spr18**	380,854	19,326	375,022	19,326	1.5	0.0
iku-spr18	31,854	20,996	27,007	20,906	15.2	-0.4
lums-fal17	358	251	349	251	2.5	0.0
mary-fal18	5,398	3,142	4,578	3,149	15.2	0.2
muni-fi-fal17	3,513	1,584	3,275	1,590	6.8	0.4
muni-fspsx-fal17	40,531	1,816	31,305	1,816	22.8	0.0
*muni-pdfx-fal17**	165,225	-	156,904	-	5.0	-
*pu-d9-fal19**	125,555	-	91,204	-	27.4	-
tg-spr18	12,704	12,504	12,704	12,378	0.0	-1.0

Table 9.3: Best solution and bound values found in 24 hours with and without solution sharing. The root node relaxation did not converge within the time limit for instances marked with an *.

of the 16 used in the competition), and six fix-and-optimize processes (each using a different neighborhood and 4 threads).

Figure 9.3 shows the solution values per time for instances *agh-fal17*, *iku-spr18*, *agh-h-spr17*, and *muni-fsps-spr17c* respectively. The search stagnates and begins diversification for the latter two instances. Note that since fix-and-optimize uses all heuristics for choosing classes while diversifying, the plots do not reflect the actual heuristic used during diversification, but keep their original format for clarity.

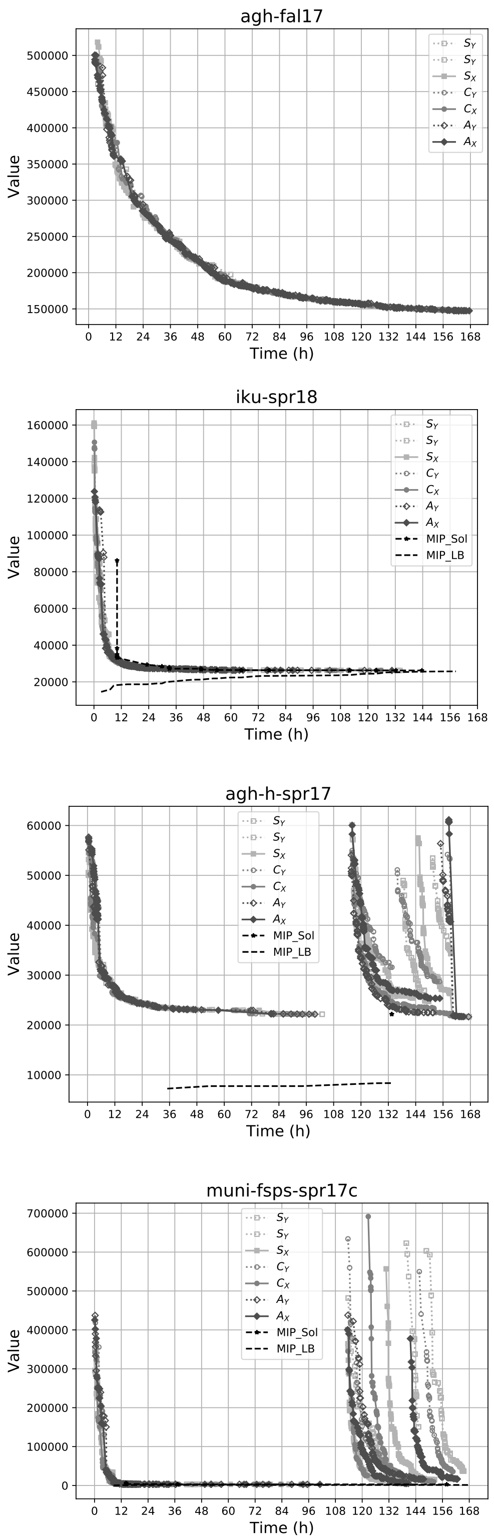


Figure 9.3: Solution values per time using the full setup and diversification for seven days on *agh-fal17*, *iku-spr18*, *agh-h-spr17* and *muni-fsps-spr17c*.

	pu-proj-fal19	*pu-d9-fal19*		
Run	LSS	PM	LSS	PM + LSS
1	178,451	50,244	89,835	46,848
2	229,073	50,715	92,014	50,356
3	230,232	50,780	89,427	53,697
4	191,685	51,405	89,977	48,767
5	218,738	47,753	84,670	48,425
Average	209,636	50,179	89,185	49,619

Table 9.4: The best solutions found for five runs of 24 hours on the *pu-proj-fal19* and *pu-d9-fal19* instances using different setups.

For *agh-h-spr17* the best solution of the initial dive is found after approximately 103 hours with an objective value of 22,162. Diversification begins 12 hours later, and at approximately 159 hours (56 hours into diversification), a new best known solution is found with an objective value of 21,919. As seen in the figure, all fix-and-optimize searches then reset to this solution and continue a collaborated search for the remaining 9 hours. In that small time span, the best known is improved 15 times to a solution with an objective value of 21,649.

The search on *muni-fsps-spr17c* also begins diversifying, but no improving solutions are found. The best solution found in the initial dive has an objective value of 2,787, and the best diversification solution has a value of 5,360. The best lower bound found by the MIP in the seven days is 1,360, resulting in a gap of 51.2% for the best found solution.

Instances *agh-fal17* and *iku-spr18* do not begin diversification in the seven day search. For *agh-fal17* the search progresses more slowly, and progressive improvement is visible in the plot until the end. No bound data is available as the MIP did not finish solving the root node linear relaxation. The search on *iku-spr18* quickly dives and then improves the best known solution in small increments throughout the search. The final best known solution has an objective value of 26,210, and the final lower bound is 25,709 giving an optimality gap of 1.91%.

9.7 Conclusion

We have proposed a parallelized matheuristic for the ITC 2019 problem. The combined approach uses multiple different methods, all MIP-based using the graph-based MIP detailed by Holm et al. (2020). Two different constructive heuristics are used to find an initial solution quickly and to find additional solutions for diversification. The fix-and-optimize matheuristic described by Mikkelsen et al. (2020) is run in parallel, and the full MIP is solved to provide information on the lower bound. The best known solution is continually shared between the fix-and-optimize searches and the MIP to push the search forward more quickly. We have also proposed a diversification scheme that is invoked when the search stagnates. Additionally, we have implemented a special setup for instances with a considerable number of students. In such cases, the parallelized matheuristic decouples class assignment and student sectioning to find initial solutions.

Computational results show that solution sharing is especially effective at helping the search move forward. Sharing solutions between six parallel fix-and-optimize searches and a single MIP solve resulted in better solutions on all instances compared to no solution sharing. The inclusion of a special setup for instances with a large student sectioning was also beneficial. This addition enables the parallelized matheuristic to find feasible solutions on instances like *pu-proj-fal19*, which otherwise proved to be too memory-intensive for the other methods used in the matheuristic. However, for the other affected instance *pu-d9-fal19* it made no difference. The diversification scheme proved successful in a seven-day search on *agh-h-spr17*, where the best known solution of the search was improved after 56 hours of diversification. For the three other tested instances, diversification was either unsuccessful or never invoked.

The proposed parallelized matheuristic has proved to be a competitive solution approach to the problem posed in the ITC 2019. It can find solutions for all instances that are part of the competition, with some even to proven optimality. The merit of the approach is further attested to by the fact that it is the winning algorithm of the ITC 2019. In Appendix 9.A, we show the objective values of our submitted solutions and bounds at the time of writing.[1]

9.7.1 Future research

Although the parallelized matheuristic proved to perform well, some immediate avenues for improvement have been identified, which should be examined in future research. For example, the diversification scheme could be improved to increase the likelihood of success. In the current implementation, the fix-and-optimize processes simply search individually and hope to find a new best known solution without much hindrance. This approach is somewhat dependent on randomness. Perhaps it would be beneficial to divide diversification into two phases. First, each fix-and-optimize would search individually and produce some high-quality solutions. In phase two, the fix-and-optimize processes start sharing solutions once more, using the potential solutions from phase one as starting points. Another interesting option is looking into applying path relinking to combine high-quality solutions and use the results for further searching. Yet another option is to dedicate a single fix-and-optimize process to generating high-quality diversification solutions from the beginning. Then, once diversification is necessary, the other fix-and-optimize processes use these solutions instead of random initial solutions. Such an approach is described for the high school timetabling problem in Saviniec et al. (2018), where it achieved state-of-the-art performance.

In our implementation, each fix-and-optimize matheuristic only uses a single neighborhood during the default "intensification" search. Solution sharing is one approach for countering problems resulting from such a rigid setting. Another option is to look into each search using multiple or all neighborhoods

[1]An updated table is available at `https://dsumsoftware.com/itc2019/`.

and then adaptively updating how often each neighborhood is used, similar to adaptive large neighborhood search (Røpke and Pisinger, 2006). That way, neighborhoods that consistently move the search forward are used more often than the underperforming neighborhoods, which should help find better solutions quicker.

We tried decomposing the problem into class assignment and student sectioning for instances with 30,000 students or more. This affected two instances: *pu-proj-fal19* and *pu-d9-fal19*. For *pu-proj-fal19* the decomposition proved to be invaluable, but made no major difference for *pu-d9-fal19*. Perhaps the number of students should not be the only deciding factor for determining if this decoupling should be done. For example, the search on *agh-fal17* was still showing moderate progression after seven days of running the full parallelized matheuristic, indicating that it is a very difficult instance to solve. This instance is also very large (has the largest number of classes besides *pu-proj-fal19*) and could perhaps benefit from the same decoupling, even though it "only" has 6,925 students. Additionally, other forms of decomposition should be examined.

9.A Our submitted solutions and bounds

Instance	Competition best	Lower bound	Gap (%)
agh-fis-spr17	3,081	1,174	61.9%
agh-ggis-spr17	35,808	23,164	35.3%
bet-fal17	290,086	89,278	69.2%
iku-fal17	18,968	18,001	5.1%
mary-spr17	14,910	14,359	4.2%
muni-fi-spr16	3,756	3,556	5.3%
muni-fsps-spr17	868	868	0.0%
muni-pdf-spr16c	36,487	14,279	60.9%
pu-llr-spr17	10,038	10,038	0.0%
tg-fal17	4,215	4,215	0.0%
agh-ggos-spr17	3,055	1,733	43.3%
agh-h-spr17	23,502	8,945	61.9%
lums-spr18	95	24	74.7%
muni-fi-spr17	3,825	2,500	34.6%
muni-fsps-spr17c	2,596	1,361	47.6%
muni-pdf-spr16	18,151	13,008	28.3%
nbi-spr18	18,014	18,014	0.0%
pu-d5-spr17	15,910	6,981	56.1%
pu-proj-fal19	148,016	13,899	90.6%
yach-fal17	1,239	516	58.4%
agh-fal17	186,200	1,999	98.9%
bet-spr18	348,589	63,444	81.8%
iku-spr18	25,878	25,781	0.6%
lums-fal17	349	253	27.5%
mary-fal18	4,423	3,496	21.0%
muni-fi-fal17	2,999	1,772	40.9%
muni-fspsx-fal17	17,074	7,747	54.6%
muni-pdfx-fal17	117,412	2,892	97.5%
pu-d9-fal19	43,006	3,302	92.3%
tg-spr18	12,704	12,704	0.0%

Table 9.5: An overview of the best solution we submitted for each instance during the competition. The reported lower bounds (and thereby gaps) might have been improved after the conclusion of the competition.

Acknowledgements

The authors have contributed equally to this work. The authors would like to thank Bernd Dammann and Sebastian Borchert for providing valuable support for working with high-performance computing. Dennis S. Holm's Ph.D. project is part of the Data Science for University Management project (dsum-software.com) funded by MaCom A/S and Innovation Fund Denmark (IFD). Rasmus Ø. Mikkelsen's industrial Ph.D. project is funded by IFD. IFD has supported the work solely financially and has not participated in any research-related activities.

References

Burke, E. K. and Petrovic, S. (2002). "Recent research directions in automated timetabling". In: *European Journal of Operational Research* 140.2, pp. 266–280.

Carter, M. and Laporte, G. (1998). "Recent developments in practical course timetabling". eng. In: *Lecture Notes in Computer Science* 1408, pp. 3–19. ISSN: 03029743.

Di Gaspero, L., Mccollum, B., and Schaerf, A. (Jan. 2007). "The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3)". In:

Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2020). "A Graph Based MIP Formulation of the International Timetabling Competition 2019". Manuscript submitted for publication.

Lewis, R. (2008). "A survey of metaheuristic-based techniques for university timetabling problems". In: *OR spectrum* 30.1, pp. 167–190.

Mikkelsen, R., Holm, D., Sørensen, M., and Stidsen, T. (2020). "A fix-and-optimize Matheuristic for the International Timetabling Competition 2019". Manuscript submitted for publication.

Müller, T., Rudová, H., and Müllerová, Z. (2018a). *University course timetabling and International Timetabling Competition 2019*. `https://www.unitime.org/present/patat18-slides.pdf`. Accessed: 2021-04-12. UniTime.

Müller, T., Rudová, H., and Müllerová, Z. (2018b). "University course timetabling and international timetabling competition 2019". In: *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria.* Ed. by E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, pp. 5–31.

Røpke, S. and Pisinger, D. (Nov. 2006). "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows". In: *Transportation Science* 40.4, pp. 455–472. DOI: `10.1287/trsc.1050.0135`. URL: `http://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0135`.

Saviniec, L., Santos, M. O., and Costa, A. M. (2018). "Parallel local search algorithms for high school timetabling problems". In: *European Journal of Operational Research* 265.1, pp. 81–98.

Schaerf, A. (1999). “A survey of automated timetabling”. In: *Artificial intelligence review* 13.2, pp. 87–127.

Tripathy, A. (1992). “Computerised decision aid for timetabling—a case analysis”. In: *Discrete applied mathematics* 35.3, pp. 313–323.

Chapter 10

Comparing Exact and Metaheuristic Methods for a Real-World Examination Timetabling Problem

Mats Carlsson[b] · Sara Ceschia[c] · Luca Di Gaspero[c] · Rasmus Ørnstrup Mikkelsen[a] · Andrea Schaerf[c] · Thomas Jacob Riis Stidsen[a]

[a]Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

[b]RISE Research Institutes of Sweden, Kista, Sweden

[c]University of Udine, Polytechnic Department of Engineering and Architecture, Via delle Scienze 206, 33100 Udine, Italy

Status: Submitted to Journal of Heuristics

Abstract: We propose a portfolio of exact and metaheuristic methods for the rich examination timetabling problem introduced by Battistutta et al. (2020). The problem includes several real-world features that arise in Italian universities, such as exams split into two parts, possible requirements of multiple rooms for a single exam, and periods and rooms unavailabilities and preferences.

We developed a CP model encoded in the MiniZinc modeling language and solved it with Gecode, as well as two MIP models solved with Gurobi. The first MIP model is encoded natively, and the second one again in MiniZinc. Finally, we extended the metaheuristic method based on Simulated Annealing of Battistutta et al. by introducing a new neighborhood relation.

We compare the different techniques on the real-world instances provided by Battistutta et al., which have been slightly refined by correcting some minor issues. All instances and solutions are publicly available, together with a solution checker, for inspection and future comparisons.

Keywords: Examination Timetabling · Constraint Programming · Integer Programming · Simulated Annealing

10.1 Introduction

The timetabling of an examination session is a difficult and crucial task that every university department has to solve regularly. Many versions of the examination timetabling problem (ETTP) have been proposed in the literature, as any educational institution has its own rules and practices. They range from very simple ones, in which only exam conflicts are taken into consideration, so that they turn out to be simple extensions of the *graph coloring* problem (see, e.g., Carter et al., 1996), to extremely complex ones that include room constraints, preferences, heterogeneous timeslots, and many other practical features (see, e.g., McCollum et al., 2007).

In this paper, we consider the real-world version introduced by Battistutta et al. (2020) that applies to Italian universities, which includes composite exams, preferences and unavailabilities for periods and rooms, exams in multiple rooms, curricula, and many other features. The original formulation of Battistutta et al. has been slightly refined to better capture a few real-world practices and to correct some minor issues.

For this problem, we propose a portfolio of solution techniques comprising both exact and metaheuristic methods. Regarding the exact techniques, we developed two Mixed Integer Programming (MIP) models and a Constraint Programming (CP) model, where the latter model is an enhanced version of the one originally proposed by Battistutta et al. On the metaheuristic side, we improved the Simulated Annealing (SA) proposed by Battistutta et al., by developing a new neighborhood relation.

Our search methods have been tested and compared among themselves and with the method by Battistutta et al., providing insights on the performances of the different methods.

All instances and best solutions are available at `https://bitbucket.org/satt/examtimetablinguniuddata` for inspection and future comparison, along with a solution checker that reports the value of the objective function of the solution. Instances are available in both JSON and `dzn` (MiniZinc data files) formats.

10.2 Problem Formulation

Our problem consists of scheduling a final exam session of a set of courses in a set of rooms within a given horizon. Battistutta et al., 2020 provide a detailed introduction to the problem, however, for self-containedness, we give a brief description here.

ID	Name	Description
H1	RoomRequest	Rooms assigned to an event must be of the correct type and quantity
H2	RoomOccupation	There must be at most one event per room per period
H3	HardConflicts	Two events in *hard conflict* must have different periods
H4	Precedences	If one event *precedes* another, the first one must be scheduled before the second one
H5	Unavailabilities	Events cannot be assigned to an unavailable period or an unavailable room
S1	SoftConflicts	Two event in *soft conflict* should have different periods
S2	Preferences	Events should not be assigned to an undesired period or an undesired room
S3	Distances	Requested time separations between events should be observed (directed and undirected)

Table 10.1: Hard and soft constraints employed in the problem formulation.

For each course, we have to schedule one or more repetitions of the same exam within the session. In turn, each exam can be composed of one or two parts (written and oral). We call each single part of an exam an *event*.

Events must be assigned to a period, which represents a timeslot and a day of the session. Events take place in rooms, which are classified, according to their size, as small, medium, or large. A single event might require more than one room, but they all must be of the same type. Events might also require no room at all[1] so that they are conventionally assigned to the *dummy* room.

Courses are grouped into curricula that determine conflicts and separations between their exams. Each curriculum has a set of primary and secondary courses, which imply different levels of conflicts and different distances among exams.

The constraints, which as customary are split into hard (H) and soft (S) ones, are summarized in Table 10.1.

10.3 Related Work

Examination timetabling is a classical optimization problem that has been extensively studied in the scientific literature see, e.g., Qu et al., 2009; Schaerf, 1999. Here, we discuss the formulations proposed, with particular attention to those equipped with some public datasets, which can be used to compare search methods.

[1]For example, because they are held at the teacher's office.

The two formulations that have received most attention in the literature are the one by Carter et al., 1996 and the one proposed by McCollum et al., 2007 as part of the 2nd International Timetabling competition (ITC2007). These two formulations are rather different from each other, as the first is an extremely simple formulation (basically an extension of graph-coloring), whereas the second one is more complex, including several peculiar rules of the British universities. Both formulations are complemented by very challenging datasets composed of 13 and 12 instances, respectively, none of which has been solved to proven optimality so far.

Another dataset comes from the Yeditepe University (Turkey) introduced by Özcan and Ersoy, 2005 and subsequently modified by Bilgin et al., 2006, who also extended the Carter dataset by including room capacity (see Parkes and Özcan, 2010 for a discussion on Yeditepe and modified Carter datasets).

A study about real-world ETTP in the Asian context was first introduced and subsequently extended by Kahar and Kendall, 2010; Kahar and Kendall, 2015, who presented the case of Universiti Malaysia Pahang (Malaysia). Demeester et al., 2012 addressed the ETTP at KAHO Sint-Lieven (Belgium), whereas Müller, 2016 showed the application of examination timetabling to a large American university (Purdue University).

Woumans et al., 2016 proposed another interesting formulation, which addressed the problem from a student-centric perspective, considering the possibility of scheduling the same exam more than once if it improves the fairness among different student groups. The authors developed a Column Generation approach applied to a test case at the KU Leuven campus (Belgium). Muklason et al., 2017 carried out an in-depth analysis of fairness in examination timetabling.

More recent activity includes Keskin et al., 2018, Abou Kasm et al., 2019, and Güler et al., 2021 that have presented different specific formulations for real-world ETTPs.

Our formulation (Battistutta et al., 2020) consists of many peculiar constraints and objectives that have not previously been addressed together in the scientific literature. These include one or more repetitions of the same exam, which, in turn, can combine a written and an oral part; exams split across multiple rooms; and primary and secondary courses for each curriculum, determining different levels of conflicts between the respective exams.

After discussing the variants of ETTPs proposed in the literature, we move onto discussing the techniques employed for their solution. Many different search methods have been applied, ranging from exact methods, heuristics, metaheuristics, and hybrid techniques.

The literature on the application of integer programming (IP) for ETTPs is relatively sparse. Some works focus on improved preprocessing and mixed integer programming (MIP) formulations of the examination timetabling problem posed in the ITC2007 (Arbaoui et al., 2015; Arbaoui et al., 2019). Others investigate university-specific problems. Al-Yakoob et al. (2010) considers both examination timetabling and proctor assignment at Kuwait University, defining and solving a MIP for each problem. Cataldo et al. (2017) solve the examination timetabling problem at Diego Portales University (Chile) using IP in four

Sets	Description
$\mathcal{E}$	The set of events
$\mathcal{P}$	The set of periods
$\mathcal{R}$	The set of rooms, including dummy room $\tilde{r}$
$\mathcal{R}^c$	The set of composite rooms ($\mathcal{R}^c \subset \mathcal{R}$)
$\mathcal{K}$	The set of room equivalence classes
$\mathcal{R}^o_{r^c}$	The set of overlapping rooms of composite room $r^c \in \mathcal{R}^c$ (member rooms and composite rooms with common member rooms)
$\mathcal{P}_e$	The set of available periods for event $e \in \mathcal{E}$
$\mathcal{R}_e$	The set of available rooms for event $e \in \mathcal{E}$
$\mathcal{K}_e$	The set of available room equivalence classes for event $e \in \mathcal{E}$
F	The set of exam pairs with precedence constraints (H4)
HC_e	The set of events that is in hard conflict with $e \in \mathcal{E}$ (H3)
Helper	**Description**
$O(\cdot)$	Ordinal operator

Table 10.2: Notation for modeling the problem.

stages. The later problem is, similar to ours, curriculum-based instead of the more common post-enrollment variant. Recently, Al-Hawari et al., 2020 implemented a multistage MIP approach to solve the ETTP at German Jordanian University (Jordan), which is validated both on real-world instances and on Carter's benchmarks.

June et al., 2019 studied the ETTP at the Universiti Malaysia Sabah Labuan International Campus (UMSLIC) and developed a hybrid solution method that integrates CP and SA in two phases: an initial feasible schedule is obtained through constraint programming, then the quality of the solution is improved by SA. The method was tested on datasets collecting the data of two semesters at UMSLIC (Malaysia). Genc and O'Sullivan, 2020 provided a CP model, mainly based on bin-packing constraints, to solve the examination timetabling problem at University College Cork (Ireland).

Metaheuristics have proven to be really effective in solving ETTPs. Current best-known results on Carter instances have been obtained by the SA developed by Bellio et al., 2021, the Genetic Algorithm of Leite et al., 2018 and the Great Deluge approach by Burke and Bykov, 2016. Similarly, state-of-the-art solvers for the ITC2007 formulation have implemented variants of SA (Burke and Bykov, 2016, Battistutta et al., 2017, and Leite et al., 2019).

10.4 Search Methods

In this section, we describe the search methods developed in our study, namely Mixed Integer Programming (Sections 10.4.1 and 10.4.2), Constraint Programming (Section 10.4.3), and Simulated Annealing (Section 10.4.4). Table 10.2 shows some common notation used in our MIP and CP models of the problem.

Variables	Description
$x_{e,p,r} \in \mathbb{B}$	1 if event $e \in \mathcal{E}$ is assigned to period $p \in \mathcal{P}_e$ and room $r \in \mathcal{R}_e$
$y_{e,p} \in \mathbb{B}$	1 if event $e \in \mathcal{E}$ is assigned to period $p \in \mathcal{P}_e$
$h_e \in \mathbb{Z}_0$	The ordinal value of the period assigned to event $e \in \mathcal{E}$

Table 10.3: Decision variables of the MIP model.

10.4.1 Mixed Integer Programming

In this section, we define an Integer Programming (IP) model for the problem. However, we can relax several of the integer variables, resulting in a Mixed Integer Programming (MIP) model. We generally refer to the model as a MIP for simplicity, although it may be an IP. We use both model variants for computational testing and explicitly state when the model is an IP model. When comparing solution approaches we designate this method **MIP**. In Section 10.4.1.1 we describe a two-stage decomposition approach using the model which we call **MIP**$_{\text{2S}}$.

Table 10.3 shows the core decision variables of the model. We have $x_{e,p,r}$ as the main decision variable, determining each event's period and room assignment. We include auxiliary variable $y_{e,p}$ to simplify writing some constraints and reduce the number of non-zeros of the model. Additionally, we include h_e to attain the ordinal value of the period assigned to each event. We use "lazy" notation and do not include the most obvious conditions in sums. For example, since we only define $x_{e,p,r}$ for $p \in \mathcal{P}_e \wedge r \in \mathcal{R}_e$, a sum over events for $x_{e,p,r}$ should be written as

$$\sum_{\substack{e \in \mathcal{E} \\ :\ r \in \mathcal{R}_e \wedge\ p \in \mathcal{P}_e}} x_{e,p,r}$$

but we simply write

$$\sum_{e \in \mathcal{E}} x_{e,p,r}$$

Constraints (10.1) - (10.7) ensure that we satisfy all hard constraints. Constraint (10.1) assigns each event to an available period and room, satisfying H1 and H5. Constraints (10.2) and (10.3) enforce H2 by ensuring that at most one event can use a single room in any period and that composite rooms can only be used if none of its overlapping rooms are simultaneously used. In constraint (10.3) M equals the number of elements in the overlapping rooms sum. We enforce precedence relationships (H4) using (10.4) and avoid hard conflicts (H3) using (10.5), where M equals the number of elements in the sum. Constraints (10.6) and (10.7) set the values of $y_{e,p}$ and h_e, respectively.

$$\sum_{p\in\mathcal{P}_e}\sum_{r\in\mathcal{R}_e} x_{e,p,r} = 1 \qquad \forall e\in\mathcal{E} \tag{10.1}$$

$$\sum_{e\in\mathcal{E}} x_{e,p,r} \leq 1 \qquad \forall r\in\mathcal{R}, p\in\mathcal{P} \tag{10.2}$$

$$M\sum_{e\in\mathcal{E}} x_{e,p,r^c} + \sum_{r^o\in\mathcal{R}^o_{r^c}}\sum_{e\in\mathcal{E}} x_{e,p,r^o} \leq M \qquad \forall r^c\in\mathcal{R}^c, p\in\mathcal{P} \tag{10.3}$$

$$h_{e_1} - h_{e_2} \leq -1 \qquad \forall (e_1,e_2)\in F \tag{10.4}$$

$$M\cdot y_{e,p} + \sum_{e_2\in HC_e} y_{e_2,p} \leq M \qquad \forall e\in\mathcal{E}, p\in\mathcal{P}_e \tag{10.5}$$

$$y_{e,p} - \sum_{r\in\mathcal{R}_e} x_{e,p,r} = 0 \qquad \forall e\in\mathcal{E}, p\in\mathcal{P}_e \tag{10.6}$$

$$\sum_{p\in\mathcal{P}_e} O(p)\cdot y_{e,p} = h_e \qquad \forall e\in\mathcal{E} \tag{10.7}$$

The problem includes three types of soft constraints. The first (S1) concerns soft conflicts, which may occur between events with a primary/secondary (PS) curriculum and secondary/secondary (SS) curriculum connection (see Battistutta et al., 2020 for details). For each event $e\in\mathcal{E}$ we define a set of PS and SS conflicting events as SC_e^{PS} and SC_e^{SS}, respectively. We also introduce integer variables $s_{e,p}^{PS}$ and $s_{e,p}^{SS}$ to respectively "count" the number of PS and SS soft conflicts for event e in period p. We set the variable values using constraints (10.8) and (10.9). For these constraints, we get the values of big-M as the number of elements in the sum, e.g., $M_{e,p}^{PS} = \left|\left\{e_2\in SC_e^{PS} : O(e) < O(e_2) \wedge p\in\mathcal{P}_{e_2}\right\}\right|$. The $O(e) < O(e_2)$ condition ensures that a conflict is only counted once. Variables $s_{e,p}^{PS}$ and $s_{e,p}^{SS}$ are bounded by 0 and $M_{e,p}^{PS}$ and $M_{e,p}^{SS}$, respectively.

$$M_{e,p}^{PS}\cdot y_{e,p} + \sum_{\substack{e_2\in SC_e^{PS}\\ :\ O(e)<O(e_2)}} y_{e_2,p} \leq s_{e,p}^{PS} + M_{e,p}^{PS} \qquad \forall e\in\mathcal{E}, p\in\mathcal{P}_e \tag{10.8}$$

$$M_{e,p}^{SS}\cdot y_{e,p} + \sum_{\substack{e_2\in SC_e^{SS}\\ :\ O(e)<O(e_2)}} y_{e_2,p} \leq s_{e,p}^{SS} + M_{e,p}^{SS} \qquad \forall e\in\mathcal{E}, p\in\mathcal{P}_e \tag{10.9}$$

To penalize violations of soft conflicts, we add the following terms to the objective function, where β^{PS} and β^{SS} denote the soft conflict costs for PS and SS event pairs, respectively.

$$Cost^{\mathrm{S1}} = \beta^{PS}\sum_{e\in\mathcal{E}}\sum_{p\in\mathcal{P}_e} s_{e,p}^{PS} + \beta^{SS}\sum_{e\in\mathcal{E}}\sum_{p\in\mathcal{P}_e} s_{e,p}^{SS}$$

The second soft constraint (S2) is event-period and event-room preferences. We define non-negative costs α_{ep} and α_{er} and add the following terms to the objective function.

$$Cost^{\mathrm{S2}} = \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} \alpha_{ep} y_{e,p} + \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} \sum_{r \in \mathcal{R}_e} \alpha_{er} x_{e,p,r}$$

As the third soft constraint (S3), the problem includes preferred distances between event pairs. For these event pairs, we are given a minimum/maximum distance parameter $P^{min}_{e_1,e_2}$ and $P^{max}_{e_1,e_2}$, respectively. First, we show how we "measure" the distance between event pairs before showing how we penalize violations. Let $DP^{\leftarrow}$ and $DP^{\leftrightarrow}$ be the sets of event pairs with a soft distance (min. or max.) constraint with directed and undirected distances, respectively. An event pair (e_1, e_2) has a directed distance requirement when we have a hard precedence constraint (H4) that guarantees that e_1 precedes e_2.

In order to include these soft constraints, we introduce integer variable d_{e_1,e_2} to take the absolute value of the distance between e_1 and e_2. Notice that the maximum distance possible between any two events is always bounded by $|\mathcal{P}|$. When event pairs have a directed distance requirement, we set d_{e_1,e_2} correctly using constraint (10.10).

$$d_{e_1,e_2} = h_{e_2} - h_{e_1} \quad \forall (e_1, e_2) \in DP^{\leftarrow} \tag{10.10}$$

When event pairs have an undirected distance requirement, we need to use some additional auxiliary variables and constraints to get the absolute distance correctly. We introduce integer variable $d^{\leftrightarrow}_{e_1,e_2}$ to measure the distance between the two events. We set this variable using constraint (10.11), and the variable may then take a negative value. We need $d_{e_1,e_2} = |d^{\leftrightarrow}_{e_1,e_2}|$, which we linearize in the following.

We introduce binary variable $g^{\leftrightarrow}_{e_1,e_2}$ to be 1 if $d^{\leftrightarrow}_{e_1,e_2}$ is positive and 0 otherwise. We set $g^{\leftrightarrow}_{e_1,e_2}$ using constraints (10.12) and (10.13). Then we introduce two variables $d^{abs_1}_{e_1,e_2}$ and $d^{abs_2}_{e_1,e_2}$ where exactly one takes the absolute value of $d^{\leftrightarrow}_{e_1,e_2}$ and the other takes the value of zero. We set the values of these two variables using (10.14) - (10.18). For simplicity in writing the model, we set the value of d_{e_1,e_2} using constraint (10.19). In practice, we (of course) simply use $d^{abs_1}_{e_1,e_2} + d^{abs_2}_{e_1,e_2}$ in place of d_{e_1,e_2}.

$$d^{\leftrightarrow}_{e_1,e_2} = h_{e_2} - h_{e_1} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.11}$$
$$d^{\leftrightarrow}_{e_1,e_2} \leq |\mathcal{P}| \cdot g^{\leftrightarrow}_{e_1,e_2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.12}$$
$$d^{\leftrightarrow}_{e_1,e_2} \geq - |\mathcal{P}| \left(1 - g^{\leftrightarrow}_{e_1,e_2}\right) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.13}$$
$$d^{abs_1}_{e_1,e_2} \leq |\mathcal{P}| g^{\leftrightarrow}_{e_1,e_2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.14}$$
$$d^{abs_1}_{e_1,e_2} \geq - |\mathcal{P}| g^{\leftrightarrow}_{e_1,e_2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.15}$$
$$d^{abs_1}_{e_1,e_2} \leq d^{\leftrightarrow}_{e_1,e_2} + |\mathcal{P}| \left(1 - g^{\leftrightarrow}_{e_1,e_2}\right) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.16}$$
$$d^{abs_1}_{e_1,e_2} \geq d^{\leftrightarrow}_{e_1,e_2} - |\mathcal{P}| \left(1 - g^{\leftrightarrow}_{e_1,e_2}\right) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.17}$$
$$d^{abs_2}_{e_1,e_2} = d^{abs_1}_{e_1,e_2} - d^{\leftrightarrow}_{e_1,e_2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.18}$$
$$d_{e_1,e_2} = d^{abs_1}_{e_1,e_2} + d^{abs_2}_{e_1,e_2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{10.19}$$

Table 10.4 shows an overview of when min. and max. distance soft constraints exist between two events and if the distance is directed. Additionally, the table shows the variable used to capture each specific distance violation, the associated cost parameter, and the event-pair sets for each category. To get the minimum and maximum distance violation we respectively use constraints with the following structure: $p_{e_1,e_2} + d_{e_1,e_2} \geq P^{min}_{e_1,e_2}$, and $d_{e_1,e_2} - p_{e_1,e_2} \leq P^{max}_{e_1,e_2}$. Thus, constraints (10.20) - (10.24) correctly set the p_{e_1,e_2} variables in the order they are shown in Table 10.4.

Event pair connection	Type	Directed	Variable	Cost	Set
Exams of same course	min	Yes	$p^{minE}_{e_1,e_2}$	γ^E	DP^E
W/O parts of same exam	min	Yes	$p^{minWO}_{e_1,e_2}$	γ^{WO}	DP^{WO}
W/O parts of same exam	max	Yes	$p^{maxWO}_{e_1,e_2}$	γ^{WO}	DP^{WO}
Courses in PP curriculum	min	No	$p^{minPP}_{e_1,e_2}$	γ^{PP}	DP^{PP}
Courses in PS curriculum	min	No	$p^{minPS}_{e_1,e_2}$	γ^{PS}	DP^{PS}

Table 10.4: Overview of minimum and maximum distance soft constraints.

$$p^{minE}_{e_1,e_2} + d_{e_1,e_2} \geq P^{min}_{e_1,e_2} \quad \forall(e_1,e_2) \in DP^E \tag{10.20}$$

$$p^{minWO}_{e_1,e_2} + d_{e_1,e_2} \geq P^{min}_{e_1,e_2} \quad \forall(e_1,e_2) \in DP^{WO} \tag{10.21}$$

$$d_{e_1,e_2} - p^{maxWO}_{e_1,e_2} \leq P^{max}_{e_1,e_2} \quad \forall(e_1,e_2) \in DP^{WO} \tag{10.22}$$

$$p^{minPP}_{e_1,e_2} + d_{e_1,e_2} \geq P^{min}_{e_1,e_2} \quad \forall(e_1,e_2) \in DP^{PP} \tag{10.23}$$

$$p^{minPS}_{e_1,e_2} + d_{e_1,e_2} \geq P^{min}_{e_1,e_2} \quad \forall(e_1,e_2) \in DP^{PS} \tag{10.24}$$

We then extend the objective function with

$$\begin{aligned} Cost^{S3} = \gamma^E \sum_{(e_1,e_2)\in DP^E} p^{minE}_{e_1,e_2} + \gamma^{WO} \sum_{(e_1,e_2)\in DP^{WO}} \left(p^{minWO}_{e_1,e_2} + p^{maxWO}_{e_1,e_2}\right) \\ + \gamma^{PP} \sum_{(e_1,e_2)\in DP^{PP}} p^{minPP}_{e_1,e_2} + \gamma^{PS} \sum_{(e_1,e_2)\in DP^{PS}} p^{minPS}_{e_1,e_2} \end{aligned}$$

Finally, we note that we can relax the soft constraint penalty counting variables ($s^{PS}_{e,p}$, $s^{SS}_{e,p}$, $p^{minE}_{e_1,e_2}$, $p^{minWO}_{e_1,e_2}$, $p^{maxWO}_{e_1,e_2}$, $p^{minPP}_{e_1,e_2}$, and $p^{minPS}_{e_1,e_2}$) to be continuous, as they naturally will attain integer values using the given constraints. Thus, we can express the model either as an IP or a MIP.

10.4.1.1 Two-stage decomposition

A typical approach for timetabling problems, especially for IP-based methods, is constructing the timetable in two stages (see, e.g., Daskalaki and Birbas, 2005; Kristiansen et al., 2015; Al-Yakoob and Sherali, 2015). One successful decomposition strategy involves first assigning events to periods and afterward to rooms (Lach and Lübbecke, 2008; Lach and Lübbecke, 2012; Sørensen and Dahms, 2014). For this decomposition scheme to be effective, we must ensure that the solution found in the first stage is feasible or that we can easily fix

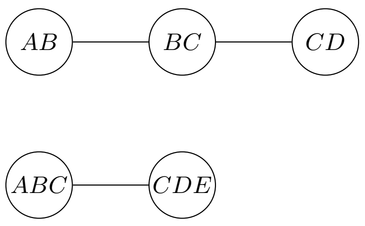


Figure 10.1: Composite room conflict graph for the given example.

any infeasibility in the second stage. In this problem, the only infeasibility that could arise between the two stages is not observing the room double-booking constraints (H2). However, rooms are generally in excess for the available data, especially since events can use rooms larger than they requested. Additionally, we have no event-room forbidding (hard) constraints in the current data, and thus, with regards to feasibility, events are indifferent to specific rooms, and rooms can be treated as generic rooms of a specific type.

Here we discuss how we implement such a decomposition for this problem, which we designated $\mathbf{MIP}_{2\mathrm{S}}$. In the first stage, $y_{e,p}$ becomes the primary assignment variable, and we leave out all room-related aspects, e.g., the $x_{e,p,r}$ variables, room double-booking constraints, and room preference objectives. We add these parts to the model in the second stage. We use the first stage solution as a warm start and do not fix any assignments, allowing the MIP solver to change period assignments freely, and, since the second stage model is the full model, we gain valid lower bounds. In the following, we define constraints to significantly reduce the risk of overbooking a period in the first stage regarding room capacity in the second stage.

Let $\mathcal{T}$ be the set of single room sizes, i.e., $\mathcal{T} = \{S, M, L\}$ where S, M, L denote small, medium and large, respectively. Composite rooms consist of multiple rooms of the same size, and we can therefore consider the combination of room size and number of rooms, which we call a "room-type". Let $\mathcal{N}_t$ be a set of "number of rooms" for a given room size t. For example, we have $\mathcal{N}_S = \{1, 2, 4\}$ if the instance data includes rooms consisting of one (single), two, and four small rooms. Then $(n, t) = (2, L)$ denotes the room-type consisting of two large rooms.

Let $\mathcal{E}^{RT}_{n,t}$ be the set of events that require a room of room-type (n, t) and $C_{n,t,p}$ the capacity of room-type (n, t) in period p. For single rooms ($n = 1$), we set the capacity to the number of single rooms of size t, or larger, available in period p. For composite rooms, we need to handle composite room member conflicts. We consider an example with five single small rooms with three $(2, S)$ and two $(3, S)$ composite rooms. Figure 10.1 shows composite room conflict graphs for the example. By inspection, we see that at most two $(2, S)$ and one $(3, S)$ can be used at a time. These limits correspond to each graph's independence number. However, due to room unavailabilities, the graph's nodes may depend on the period p. Thus, we get the capacity for composite room-types, as the independence number of the conflict graph in period p.

Exams requesting a single room can use rooms of that size or larger, e.g., an exam requesting a small room can use small, medium, and large rooms. Exams requesting a composite room can only use composite rooms that meet their exact specification. Constraint (10.25) ensures that we observe single room occupation and (10.26) specifically limits the number of events assigned in a single period that require composite rooms.

$$\sum_{\substack{t' \in \mathcal{T} \\ : t' \geq t}} \sum_{n \in \mathcal{N}_t} n \sum_{e \in \mathcal{E}^{RT}_{n,t}} y_{e,p} \leq \sum_{\substack{t' \in \mathcal{T} \\ : t' \geq t}} C_{t,1,p} \quad \forall t \in \mathcal{T}, p \in \mathcal{P} \tag{10.25}$$

$$\sum_{e \in \mathcal{E}^{RT}_{n,t}} y_{e,p} \leq C_{n,t,p} \quad \forall t \in \mathcal{T}, n \in \mathcal{N}_t : n > 1, p \in \mathcal{P} \tag{10.26}$$

These constraints do not consider composite room conflicts for composite rooms with a different number of member rooms. However, in most cases, they are sufficient as few instances have composite rooms with three and four member rooms, and fewer still with the possibility of conflicts between them. We have not experienced a solution from the first stage being infeasible in the second stage throughout our testing. By far, it is most important to forbid double-booking between single and composite rooms with two member rooms, as they are most common. Constraints (10.25) and (10.26) handle this fully.

10.4.2 Mixed Integer Programming with MiniZinc

In this section, we define **MIP**$^{\text{MZ}}$, another MIP model for the problem. It is encoded in the MiniZinc modeling language (see Nethercote et al., 2007) and solved with the Gurobi backend (see Gurobi Optimization, LLC, 2021). MiniZinc has rich support for binarizing multiple-value domains and linearizing constraints to make models suitable for execution by MIP solvers. We have not relied much on that support; instead, our model uses 0-1 variables and linear constraints almost exclusively. In fact, **MIP**$^{\text{MZ}}$ was derived from the **CP** model (see Section 10.4.3) by manual binarization and linearization.

As noted in Section 10.4.1.1, events are indifferent to specific rooms, and rooms can be treated as generic rooms of a specific type, so we say that rooms r_1 and r_2 are *equivalent* iff they are interchangeable wrt. constraints H1, H5, and S2. We split $\mathcal{R}$ into a set $\mathcal{K}$ of equivalence classes. Then, we solve the problem in terms of equivalence classes instead of specific rooms, relaxing constraint H2 to allow multiple events per class simultaneously up to the capacity of the class. Finally, we trivially transform the obtained solution back into one expressed in terms of specific rooms.

We now present the MiniZinc model step by step.

These parameters correspond to $|\mathcal{E}|, |\mathcal{P}|, |\mathcal{R}|, |\mathcal{K}|$:

```
int: Events;
int: Periods;
int: Rooms;
int: CRooms = length(RoomClasses);
```

The main decision variables are as follows. `EventPeriodCRoom` is the main decision variable that relates events, periods, and room classes. `EventPeriod` is an auxiliary function giving the period as a function of the given event.

```
array [1..Events,1..Periods,1..CRooms] of var 0..1: EventPeriodCRoom;
var 1..Periods: EventPeriod(int: e) =
  sum(p in 1..Periods, r in 1..CRooms)(p * EventPeriodCRoom[e,p,r]);
```

Another decision variable `CostDistance` facilitates the part of the objective function that depends on the temporal distance between pairs of events. We have transformed the raw input data, which was given as several large arrays and several cost terms for many pairs of events, into two arrays `CostKeys` and `CostVectors` that aggregate this data into at most one cost term per event pair. This transformation was done in the same spirit as tabling (see Dekker et al., 2017). Let i be a row of `CostKeys` containing a pair (e_1, e_2) of events and let δ be their temporal distance. Then `CostVectors`$[i, \delta]$ is the incurred cost for that pair, and `CostDistance`$[i, d] = 1$ if $d = \delta$ and 0 otherwise:

```
set of int: CostIndex = index_set_1of2(CostKeys);
set of int: Distance  = index_set_2of2(CostVectors);
array [CostIndex,Distance] of var 0..1: CostDistance;
```

The minimization statement is as follows, where the sum of the four $*$`Cost` terms forms the objective function:

```
var int: obj =  sum(ConflictDistanceCost) +
                sum(CRoomPreferenceCost) +
                sum(PeriodPreferenceCost) +
                sum(CRoomPeriodCost);

solve
  minimize obj;
```

The first MiniZinc constraint ensures that each event is assigned to exactly one period and room class:

```
constraint
  forall(e in 1..Events)(sum(EventPeriodCRoom[e,..,..]) = 1);
```

The next MiniZinc constraints eliminate forbidden event-period, event-room, and room-period combinations (constraint H1, H5):

```
% FORBIDDEN EVENT*PERIOD CONSTRAINT
constraint
  forall(e in 1..Events, p in 1..Periods, r in 1..CRooms
         where EventPeriodConstraints[e, p] = -1)
    (EventPeriodCRoom[e,p,r] = 0);

% FORBIDDEN EVENT*ROOM CONSTRAINT
constraint
  forall(e in 1..Events, p in 1..Periods, r in 1..CRooms
         where EventCRoomConstraints[e, r] = -1)
    (EventPeriodCRoom[e,p,r] = 0);

% FORBIDDEN ROOM*PERIOD CONSTRAINT
constraint
  forall(e in 1..Events, p in 1..Periods, r in 1..CRooms
         where CRoomPeriodConstraints[r, p] = -1)
    (EventPeriodCRoom[e,p,r] = 0);
```

The next MiniZinc constraint ensures that room class capacities are not exceeded (constraint H2):

```
constraint
  forall(p in 1..Periods, r in 1..CRooms-DummyRoom)(
    sum(EventPeriodCRoom[..,p,r]) <= CRoomCap[r]
   );
```

The next MiniZinc constraint ensures that a composite room cannot be used simultaneously with an overlapping room (constraint H2):

```
constraint
  forall(p in 1..Periods,
         r1 in 1..CRooms,
         r2 in 1..CRooms where r1<r2 /\ CRoomsetOverlap[r1,r2]=1)(
    min(sum(EventPeriodCRoom[..,p,r1]), sum(EventPeriodCRoom[..,p,r2])) = 0
  );
```

The next MiniZinc constraint encodes the hard conflicts rule (constraint H3), where each $c \in$ `AllCliques` is a maximal clique of events that are in hard conflict with each other. The cliques are computed by a standard max clique algorithm in a preprocessing step:

```
constraint
  forall(Clique in AllCliques, p in 1..Periods)(
    sum(e in Clique, r in 1..CRooms)(EventPeriodCRoom[e,p,r]) <= 1
  );
```

The next MiniZinc constraint enforces precedences (constraint H4):

```
constraint
  forall(e1 in 1..Events, e2 in 1..Events
         where e1<e2 /\
         Precedence[e1,e2] = 1)(EventPeriod(e1) < EventPeriod(e2)
  );
```

The next MiniZinc constraints define the `ConflictDistanceCost` term of the objective function (constraints S1 and S3). The constraints use the `CostDistance` variable (see above):

```
constraint
    forall(i in CostIndex, p in 1..Periods, q in 1..Periods
           where CostVectors[i, q-p]>0)(
      let {int: e1 = CostKeys[i,1], int: e2 = CostKeys[i,2] } in
        CostDistance[i,q-p] >=
         sum(EventPeriodCRoom[e1,p,..]) + sum(EventPeriodCRoom[e2,q,..]) - 1
      );

constraint
  forall(i in CostIndex)(
    sum(CostDistance[i,..]) <= 1
  );

constraint
  forall(i in CostIndex)(
    ConflictDistanceCost[i] =
      sum(j in Distance)(CostVectors[i,j] * CostDistance[i,j])
```

The last MiniZinc constraints define the remaining terms of the objective function (constraint S2):

```
% ROOM PREFERENCE COST
constraint
  forall(e in 1..Events, r in 1..CRooms)(
    CRoomPreferenceCost[e] >= sum(p in 1..Periods)(
      EventCRoomConstraints[e,r] * EventPeriodCRoom[e,p,r]
    )
  );

% PERIOD PREFERENCE COST
constraint
  forall(e in 1..Events, p in 1..Periods)(
    PeriodPreferenceCost[e] >= sum(r in 1..CRooms)(
      EventPeriodConstraints[e,p] * EventPeriodCRoom[e,p,r]
    )
  );

% PERIOD DEPENDENT UNDESIRED ROOMS
constraint
  forall(e in 1..Events, r in 1..CRooms)(
    CRoomPeriodCost[e] >= sum(p in 1..Periods)(
      CRoomPeriodConstraints[r,p] * EventPeriodCRoom[e,p,r]
    )
  );
```

10.4.3 Constraint Programming

In this section, we define **CP**, a Constraint Programming (CP) model for the problem. The **MIP**$^{\mathrm{MZ}}$ model, described in the previous Section 10.4.2, was derived from **CP**. Therefore, we do not repeat some details and citations already given in that section, in particular the abstraction of rooms into equivalence classes. The **CP** model is encoded in the MiniZinc modeling language (see Nethercote et al., 2007) and solved with the Gecode backend (see Gecode Team, 2019). Although CP is an exact method, we actually use a heuristic search method, because exhaustive search does not work well for this problem. The heuristic search uses restarts and Large Neighborhood Search (LNS, see Shaw, 1998; Dekker et al., 2018) on top of a branch-and-bound scheme. This is expressed by search annotations, which are passed to Gecode for execution. It is worth noting that this is supported by MiniZinc straight out of the box. We now present the MiniZinc model step by step.

As in the previous model, these parameters correspond to $|\mathcal{E}|, |\mathcal{P}|, |\mathcal{R}|, |\mathcal{K}|$:

```
int: Events;
int: Periods;
int: Rooms;
int: CRooms = length(RoomClasses);
```

The decision variables are as follows, where `CRoomPeriodIndex`(k, p) is a bijective function of the room class k and period p. Also, `CRoomPeriodIndex`(k, p) is monotonically increasing in the absolute distance between p and the middle period. In other words, the farther away p is from the middle period, the larger the function value is. `EventCRP`$[e]$ is an auxiliary variable for the (period, room class) combination of event e.

```
array [1..Events] of var 1..Periods: EventPeriod;
array [1..Events] of var 1..CRooms: EventCRoom;
array[1..Events] of var int: EventCRP =
  [CRoomPeriodIndex(EventCRoom[e],EventPeriod[e]) ::domain | e in 1..Events];
```

The first MiniZinc constraints encode constraints H1 and H5:

```
constraint
  forall(e in 1..Events, p in 1..Periods
         where EventPeriodConstraints[e, p] = -1) (EventPeriod[e] != p);

constraint
  forall(e in 1..Events, r in 1..CRooms
         where EventCRoomConstraints[e, r] = -1) (EventCRoom[e] != r);
```

By eliminating up front infeasible values from the domain of `EventCRP`, the next MiniZinc constraint contributes to encoding constraint H5:

```
constraint
  forall(e in 1..Events, c in 1..CRooms, p in 1..Periods
         where CRoomPeriodConstraints[c,p] = -1)
    (EventCRP[e] != CRoomPeriodIndex(c,p));
```

The next MiniZinc constraint encodes constraint H2:

- The quantity `count`$[k, p]$ counts the number of events for given equivalence class k and period p by means of a `global_cardinality_closed` constraint (see Stuckey et al., 2020), and its value can be at most the cardinality of given equivalence class k.

- A composite room cannot be used simultaneously with an overlapping room.

```
constraint
  let {array[1..CRooms,1..Periods] of var 0..max(CRoomCap): count} in
    global_cardinality_closed([ EventCRP[e]
                              | e in 1..Events where RoomedEvent[e]=1
                              ],
                              1..CRooms*Periods,
                              array1d(count)) /\
    forall(p in 1..Periods, r in 1..CRooms)(count[r,p] <= CRoomCap[r]) /\
    forall(p in 1..Periods,
           r1 in 1..CRooms,
           r2 in 1..CRooms where r1<r2 /\ CRoomsetOverlap[r1,r2]=1)(
        min(count[r1,p], count[r2,p]) = 0
    );
```

The next MiniZinc constraint encodes the hard conflicts rule (constraint H3), like in **MIP**$^{\mathrm{MZ}}$, but using an `alldifferent` constraint (see Stuckey et al., 2020) instead of a linear constraint:

```
constraint
  forall(c in AllCliques)(alldifferent(e in c)(EventPeriod[e]));
```

The next MiniZinc constraint enforces precedences (constraint H4), exactly as in **MIP**$^{\mathrm{MZ}}$.

```
constraint
  forall(e1 in 1..Events - 1, e2 in e1 + 1..Events
         where Precedence[e1, e2] = 1)
    (EventPeriod[e1] < EventPeriod[e2]);
```

The next MiniZinc constraint defines the `ConflictDistanceCost` term of the objective function (constraints S1 and S3), using exactly the same transformation from raw input data as in **MIP**$^{\mathrm{MZ}}$:

```
constraint
  ConflictDistanceCost =
    sum(i in index_set_1of2(CostKeys))(
      let {int: e1 = CostKeys[i,1],
           int: e2 = CostKeys[i,2],
           var int: delta = EventPeriod[e2]-EventPeriod[e1],
      } in CostVectors[i, delta]
    );
```

The last MiniZinc constraints define the remaining terms of the objective function (constraint S2):

```
constraint
  CRoomPreferenceCost =
    max(0,sum(e in 1..Events)(EventCRoomConstraints[e, EventCRoom[e]]));

constraint
  PeriodPreferenceCost =
    max(0,sum(e in 1..Events)(EventPeriodConstraints[e, EventPeriod[e]]));

constraint
  CRoomPeriodCost =
    max(0,sum(e in 1..Events)(CRoomPeriodConstraints[EventCRoom[e],
                                                     EventPeriod[e]]));
```

Finally, the search and minimization statement is as follows, where `SortedEvents` is the sequence of events in some heuristic order, and the four ∗`Cost` expressions are the terms of the objective function. The search annotations are:

- `restart_luby(100)` — the k^{th} restart is given a node limit of $100 \cdot L_k$, where L is the Luby restart sequence (see Luby et al., 1993).
- `relax_and_reconstruct(CRPE, 97)` — at every restart after finding the first solution, an LNS step is performed where 3% randomly selected decision variables are set free and the remaining 97% keep their current values.
- `int_search(CRPE, dom_w_deg, indomain_split)` — this annotation determines the order in which variables and their values are explored. The next variable to explore is the `EventCRP` variable with the smallest current domain size divided by weighted degree, breaking ties by the `SortedEvents` order. For the designated variable, explore the lower half of its domain first, splitting domains until one value has been singled out. By the monotonicity property of `CRoomPeriodIndex`, this has the effect of attempting to place events close to the middle period before attempting to place them farther away from the middle.

The choice of Luby as opposed to other restart schemes and the parameter values was made after preliminary experimentation:

```
array[1..Events] of 1..Events: SortedEvents;

array[int] of var int: CRPE = [EventCRP[e] | e in SortedEvents];

solve
  :: restart_luby(100)
  :: relax_and_reconstruct(CRPE, 97)
  :: int_search(CRPE, dom_w_deg, indomain_split)
  minimize ConflictDistanceCost + CRoomPreferenceCost +
           PeriodPreferenceCost + CRoomPeriodCost;
```

10.4.4 Simulated Annealing

The Simulated Annealing (**SA**) approach is an extension of the one proposed by Battistutta et al. (2020). In detail, to represent a state in the search space we use two vectors that store the period and the room of each event, respectively. Only periods in $\mathcal{P}_e$ and rooms in $\mathcal{R}_e$ can be assigned to event e, thus explicitly enforcing constraints H1 and H5. Conversely, the other constraints, namely H2 (RoomOccupation), H3 (HardConflicts), and H4 (Precedences) can be violated and are included in the cost function with a high weight. The cost function is thus a linear combination of the soft constraints S1–S3 and a measure of the *distance to feasibility*, corresponding to the degree of violation of constraints H2, H3, and H4.

The initial solution is generated totally at random, except that it always satisfies constraints H1 and H5. This is obtained simply by drawing randomly period and room assignments for event e from $\mathcal{P}_e$ and $\mathcal{R}_e$, respectively.

Regarding the neighborhood relation, Battistutta et al. employed the one, called MEE (MoveEventOrExam), that moves either a single event (with probability $1 - p_b$) or the two events associated with a composite exam jointly (with probability p_b).

We also use MEE, but in combination with a new neighborhood, called SE (SwapEvents), which swaps period and room of two single events. Only events that do not belong to a composite exam are included in the SE neighborhood.

At each iteration a random move is drawn from the neighborhood MEE $\cup$ SE. The move selection is biased on the basis of a parameter p_s (called *swap rate*), so that a SE move is selected with probability p_s, and a MEE move with probability $1 - p_s$. The parameters p_b and p_s are fixed experimentally, according to the tuning procedure. The drawing of the specific move inside the selected neighborhood is made according to a uniform distribution.

As customary for SA, we use the Metropolis acceptance criterion: A move is always accepted if it is improving or sideways (i.e., same cost), whereas it is accepted based on a time-decreasing exponential distribution $e^{-\Delta/T}$ in case it is worsening, where Δ is the difference of total cost induced by the move, and T is the *temperature*.

The temperature starts at the initial value T_0 and is decreased during the search by multiplying it by a value α (with $0 < \alpha < 1$) after a fixed number of samples N_s have been drawn, or a fixed number of moves N_a have been accepted. The temperature evolves according to the standard geometric cooling scheme of SA.

For the tuning procedure, we decided to use as stop criterion the total number of iterations $\mathcal{I}$, in order to keep the running time approximately equal for all configurations of the parameters. In our experiments, we fixed $\mathcal{I} = 10^8$, corresponding to an average time of approximately 660 seconds per run.

The tuning procedure has been performed using the tool JSON2RUN (Urli, 2013), which samples the configurations using the *Hammersley point set* (Hammersley and Handscomb, 1964) and implements the F-Race procedure (Birattari et al., 2010) for comparing them.

The winning configuration turned out to be: $T_0 = 188.89$, $\alpha = 0.875$, $N_s = 2092772$, $N_a = 292988$, $p_b = 0.967$, and $p_s = 0.288$.

For the experiments in the comparison with the other techniques, in order to have the same fixed running time on all instances, we use an additional stop criterion based on total running time.

10.5 Experimental Analysis

We first introduce the dataset employed in the analysis (Section 10.5.1), then we show the settings used for the comparison (Section 10.5.2), and finally, we report and discuss the experimental results.

10.5.1 Problem Instances

The dataset is composed of real-world instances extracted from various Italian universities and collected by Battistutta et al. They are written in JSON file format but are also made available in dzn format, thanks to a preprocessing procedure that splits courses into single events and distributes constraints accordingly.

In reference to the extraction and preprocessing procedures, we have corrected a few minor issues in the code of Battistutta et al. so that the instances that we use here are actually slightly revised with respect to the ones used by Battistutta et al.

The repository `https://bitbucket.org/satt/examtimetablinguniuddata` contains both the original instances and the revised ones. In addition, the it also contains a *software toolbox*, written in Python, that allow to check the validity of instances and solutions. The software has been written independently from the solvers so to be used as a sort of *third-party* solution checker in the spirit of the methodology outlined in (Bonutti et al., 2012). Besides allowing the debugging of the different solution approaches presented in this paper, this toolbox will provide against misinterpretations of the different constraints in case of future works on the same problem by different researchers, thus consenting the comparability of results. In addition, the software provides a format translator between the original (and richer) JSON format to the event based `dzn` format and a tool for computing a set of features from both the instances and the solutions.

Related to this last functionality, Table 10.5 shows the main features of the revised instances in terms of the total number of courses, events, periods, single rooms and composite rooms, and the number of timeslots (i.e., periods in a day). The name of each instance follows this pattern: Dx-y-z, where x is the department identifier, y is the exam session, and z is the academic year. It can be noticed that instances of the same department are quite homogeneous, except for D3, D4, and D5, where the number of events and periods changes during the year depending on the season of the session.

10.5.2 Setting and Tuning

We run all experimental tests in a high-performance computing cluster on 64bit computers running Scientific Linux 7.7. We use computers equipped with 756GB RAM and two Intel Xeon Gold 6226R CPUs clocked at 2.90GHz. We run the MiniZinc model using MiniZinc version 2.5.5 (see Nethercote et al., 2007) and Gecode version 6.3.0 (see Gecode Team, 2019). To solve MIP models, we use Gurobi 9.1.0 (see Gurobi Optimization, LLC, 2021). The Simulated Annealing procedure has been implemented in C++ and compiled using GNU g++ (v. 9.2.0).

For all tests, except for lower bounds (LB), we have ten runs using a single thread for an hour. To get lower bounds, we have five runs using four threads and a runtime of 24 hours. Table 10.6 shows the parameter settings used for each test. The Gurobi Presolve parameter controls the level of presolve, and a value of 2 indicates "aggressive" and 0 turns it off. The MIPFocus modifies

Instance	Courses	Events	Periods	Timeslots	Single rooms	Composite rooms
D1-1-16	261	261	40	2	64	0
D1-1-17	247	247	46	2	65	0
D1-2-16	254	254	36	2	64	0
D1-2-17	281	281	38	2	65	0
D1-3-16	239	239	26	2	65	0
D1-3-17	254	254	34	2	65	0
D1-3-18	258	258	52	2	64	0
D2-1-18	57	62	156	6	0	0
D2-2-18	58	61	162	6	0	0
D2-3-18	58	61	204	6	0	0
D3-1-16	81	164	188	4	14	3
D3-1-17	89	177	188	4	15	3
D3-1-18	87	174	188	4	15	3
D3-2-16	76	78	48	4	14	3
D3-2-17	87	88	48	4	15	3
D3-2-18	82	84	48	4	15	3
D3-3-16	78	80	48	4	14	3
D3-3-17	84	85	48	4	15	3
D3-3-18	81	83	48	4	15	3
D4-1-17	234	361	80	2	34	0
D4-1-18	223	476	80	2	34	0
D4-2-17	226	482	88	2	34	0
D4-2-18	238	514	86	2	34	0
D4-3-17	223	235	38	2	34	0
D4-3-18	240	260	38	2	34	0
D5-1-17	134	277	122	2	17	4
D5-1-18	148	311	136	2	20	4
D5-2-17	125	344	136	2	17	4
D5-2-18	156	426	122	2	20	4
D5-3-18	129	132	24	2	17	4
D6-1-16	189	487	66	2	29	41
D6-1-17	194	494	80	2	29	41
D6-1-18	198	511	80	2	29	41
D6-2-16	193	511	78	2	29	41
D6-2-17	195	501	88	2	29	41
D6-2-18	207	539	90	2	29	41
D6-3-16	192	346	58	2	29	41
D6-3-17	192	350	52	2	29	41
D7-1-17	63	150	155	5	22	0
D7-2-17	60	136	330	10	22	0

Table 10.5: Instance features.

Method	Parameters
SA	$T_0 = 188.89$, $\alpha = 0.875$, $N_s = 6278316$, $N_a = 878964$, $p_b = 0.967$, and $p_s = 0.288$.
CP	Default
MIP$^{\mathrm{MZ}}$	MIPFocus = 1
MIP	Presolve = 2, MIPFocus = 1
MIP$_{2\mathrm{S}}$	Stage 1: Presolve = 2, MIPFocus = 1 Stage 2: Presolve = 0, MIPFocus = 1
LB	Presolve = 2, MIPFocus = 3

Table 10.6: Parameters used for testing.

Gurobi's high-level search strategy, such that a value of 1 instructs Gurobi to focus on finding solutions and 3 to focus on improving the bound.

Through testing we found that when running **MIP** (the full model discussed in Section 10.4.1), we get better results when using the MIP variant of the model as opposed to the IP variant. The **MIP**$_{2\mathrm{S}}$ gets better results when using the IP. Additionally, testing revealed that the best time distribution for **MIP**$_{2\mathrm{S}}$ was to run the first stage for 99% of the available time, leaving just 36 seconds for the second stage. It is generally easy for the solver to add rooms in the second stage warm-start without incurring in any room preference penalties. Thus time is better spent in the first stage. We skip presolving as there is no benefit given such a short time limit, and it is better to try and eliminate any introduced room preference penalties. Finally, we use the IP model variant for getting lower bounds, as this model generally finds better bounds.

When running **MIP**$^{\mathrm{MZ}}$, we leave the presolve parameter to the default value, as an aggressive presolve strategy on this model leads to a very long presolving phase on most instances, resulting in worse performance.

For **SA**, we used the winning configuration of parameters shown in Section 10.4.4, except for N_s and N_a, which have been increased in order to have one hour running time.

10.5.3 Comparative Results

Table 10.7 reports the average results of ten runs of each technique with a timeout of one hour. The second column represents the results of the code by Battistutta et al., 2020, which we rerun on this test computer with the same timeout.

First, we notice that **SA** improves upon the previous version on all instances but two, although the gap is relatively small. The results of **CP** are somewhat inferior, with some cases in which they are particularly poor. The situation is more extreme for the other MiniZinc-based model **MIP**$^{\mathrm{MZ}}$, which has some good results, but in other cases, they are extremely bad or even with no solution returned within the given timeout. This behavior was expected as MiniZinc is a general-purpose, high-level language, which requires much less programming effort than implementing the search method from scratch. Regarding the **MIP** and **MIP**$_{2S}$ methods, unsurprisingly, the two-stage method **MIP**$_{2S}$ has (generally) better performance than the single-stage one **MIP**, as, like **CP**, it makes use of a preprocessing step that removes one dimension of the problem (i.e., the rooms), which has little effect on solution value. Finally, we notice that **MIP**^{MZ} and **MIP**$_{2S}$ provide more robust results than **SA** on two instances, namely D4-1-17 and D6-3-17, consistently obtaining the best values (276 and 30, respectively).

Regarding the lower bounds, we see that, besides the cases in which there is a perfect solution (cost 0), they are quite tight (below 20%) only in four cases. In particular, in one case the lower bound is equal to the best solution, thus proving its optimality.

10.6 Conclusions and Future Work

We have investigated three independent optimization paradigms, namely CP, MIP, and SA, for the real-world examination timetabling problem proposed by Battistutta et al. (2020).

For MIP, we have developed three different versions (**MIP**^{MZ}, **MIP**, and **MIP**$_{2S}$), thus resulting in five total alternative search methods that we compared among themselves and with the original SA of Battistutta et al. Even though the techniques have been developed independently (and by different authors), they have been run on the same machine with the same timeout, to have a fair competition ground. In addition, we also computed some valid lower bounds.

Unsurprisingly, the metaheuristic approach, properly tuned, turned out to work better on the majority of instances, but the exact techniques are close and actually better on a few instances.

All instances and solutions are available for inspection and future comparison. Furthermore, to encourage future contributions from other researchers, we have also developed a publicly available solution checker to protect against possible misinterpretations of the data and the constraints. Thanks to this tool, we have also detected and corrected some discrepancies between the JSON and dzn formats in the original files of Battistutta et al.

In the future, we will investigate the correlation of the results of the different search methods with the features of the specific instances. To this aim, we plan to collect new real-world instances and possibly develop a principled instance generator to study such correlation on a potentially unlimited number of instances.

In addition, we plan to develop hybrid techniques that could benefit from the respective advantages of the different search methods. These hybrid techniques would range from simply using SA for warm-starting MIP and CP, to more complex interaction mechanisms such as Benders (1962) decomposition or the *matheuristic* paradigm (see Maniezzo et al., 2021).

Instance	**SA**	Batt. et al.	**CP**	**MIP**$^{\text{MZ}}$	**MIP**	**MIP**$_{\text{2S}}$	Best	LB
D1-1-16	**385.0***	393.9	414.2	388.0	399.0	394.0	381	192
D1-1-17	**320.3***	328.2	370.9	323.0	329.0	324.0	318	181
D1-2-16	**524.6***	535.0	561.8	543.0	554.4	536.0	521	216
D1-2-17	**613.0***	625.7	661.9	690.0	654.0	631.0	609	233
D1-3-16	**724.9***	733.5	757.9	767.4	741.0	729.0	720	218
D1-3-17	**614.2***	622.3	648.3	634.0	639.0	621.0	612	221
D1-3-18	**265.0***	266.8*	279.6	270.0	272.0	266.0	264	190
D2-1-18	427.6*	**426.8***	440.8	464.0	448.0	N/A	426	4
D2-2-18	**22.0***	**22.0***	27.4*	N/A	**22.0***	N/A	22	20
D2-3-18	**22.0***	**22.0***	**22.0***	N/A	**22.0***	N/A	22	10
D3-1-16	**0.0***	**0.0***	1.2*	**0.0***	**0.0***	**0.0***	0	0
D3-1-17	**0.0***	**0.0***	0.6*	**0.0***	**0.0***	**0.0***	0	0
D3-1-18	**0.0***	**0.0***	0.2*	**0.0***	**0.0***	**0.0***	0	0
D3-2-16	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D3-2-17	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D3-2-18	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D3-3-16	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D3-3-17	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D3-3-18	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D4-1-17	276.6*	278.8*	296.3	**276.0***	278.0	**276.0***	276	256
D4-1-18	**1,037.3***	1,060.1	1,174.2	7,984.1	1,322.5	1,156.1	1,028	409
D4-2-17	**1,126.0***	1,150.8	1,320.7	-	1,533.9	1,490.6	1,105	459
D4-2-18	**1,594.5***	1,616.8	2,360.6	-	-	2,099.6	1,579	530
D4-3-17	**373.6***	381.1	403.3	381.0	383.6	377.0	372	224
D4-3-18	**677.4***	691.9	757.1	826.0	802.0	752.0	670	265
D5-1-17	**157.0***	158.1*	198.1	1,261.0	218.0	160.2	156	0
D5-1-18	**36.6***	38.2*	99.2	327.4	80.2	38.0	36	0
D5-2-17	**60.0***	**60.0***	81.6	-	296.0	64.0	60	0
D5-2-18	**271.6**	272.2*	339.3	-	629.2	356.0	264	0
D5-3-18	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	**0.0***	0	0
D6-1-16	443.3	**442.0***	891.2	23,194.0	1,266.2	519.8	432	61
D6-1-17	**367.1***	377.0	621.8	-	577.0	637.0	360	83
D6-1-18	**400.2***	405.7*	1,019.2	-	565.6	493.0	392	75
D6-2-16	**516.1***	527.2	1,792.9	-	741.9	573.4	506	76
D6-2-17	**564.8**	569.8*	1,667.9	-	1,095.3	715.1	558	80
D6-2-18	**202.9***	214.6	590.9	-	-	280.8	199	56
D6-3-16	**27.0***	**27.0***	34.1	**27.0***	**27.0***	**27.0***	27	27
D6-3-17	30.2*	30.3*	43.5	**30.0***	**30.0***	**30.0***	30	26
D7-1-17	**395.7***	397.5*	479.3	484.2	524.2	435.0	386	40
D7-2-17	**764.2**	764.4*	847.8	6,161.2	949.4	938.0	758	10

Table 10.7: Comparative results and lower bounds. Results in bold indicate best average solution values. Values marked with ($*$) show that the given method found the best solution for that instance.

Acknowledgements

Rasmus Ø. Mikkelsen's industrial Ph.D. project is funded by Innovation Fund Denmark (IFD). IFD has supported the work solely financially and has not participated in any research-related activities.

References

Abou Kasm, O., Mohandes, B., Diabat, A., and El Khatib, S. (2019). "Exam timetabling with allowable conflicts within a time window". In: *Computers & Industrial Engineering* 127, pp. 263–273.

Arbaoui, T., Boufflet, J.-P., and Moukrim, A. (2015). "Preprocessing and an improved MIP model for examination timetabling". In: *Annals of Operations Research* 229.1, pp. 19–40.

Arbaoui, T., Boufflet, J.-P., and Moukrim, A. (2019). "Lower bounds and compact mathematical formulations for spacing soft constraints for university examination timetabling problems". In: *Computers & Operations Research* 106, pp. 133–142.

Battistutta, M., Ceschia, S., De Cesco, F., Di Gaspero, L., Schaerf, A., and Topan, E. (2020). "Local Search and Constraint Programming for a Real-World Examination Timetabling Problem". In: *17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2020)*. Ed. by E. Hebrard and N. Musliu. Vol. 12296. LNCS. Springer International Publishing, pp. 69–81.

Battistutta, M., Schaerf, A., and Urli, T. (2017). "Feature-based tuning of single-stage simulated annealing for examination timetabling". In: *Annals of Operations Research* 252.2, pp. 239–254. ISSN: 0254-5330.

Bellio, R., Ceschia, S., Di Gaspero, L., and Schaerf, A. (2021). "Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling". In: *Computers and Operations Research* 132, pp. 1–9. ISSN: 0305-0548.

Benders, J. F. (1962). "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische mathematik* 4.1, pp. 238–252.

Bilgin, B., Özcan, E., and Korkmaz, E. E. (2006). "An experimental study on hyper-heuristics and exam timetabling". In: *International Conference on the Practice and Theory of Automated Timetabling*. Springer, pp. 394–412.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). "F-race and iterated F-race: An overview". In: *Experimental methods for the analysis of optimization algorithms*. Berlin: Springer, pp. 311–336.

Bonutti, A., De Cesco, F., Di Gaspero, L., and Schaerf, A. (2012). "Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results". In: *Annals of Operations Research* 194.1, pp. 59–70. DOI: 10.1007/s10479-010-0707-0.

Burke, E. K. and Bykov, Y. (2016). "An adaptive flex-deluge approach to university exam timetabling". In: *INFORMS Journal on Computing* 28.4, pp. 781–794.

Carter, M. W., Laporte, G., and Lee, S. Y. (1996). “Examination Timetabling: Algorithmic Strategies and Applications”. In: *Journal of the Operational Research Society* 74, pp. 373–383.

Cataldo, A., Ferrer, J.-C., Miranda, J., Rey, P. A., and Sauré, A. (2017). “An integer programming approach to curriculum-based examination timetabling”. In: *Annals of Operations Research* 258.2, pp. 369–393.

Daskalaki, S. and Birbas, T. (2005). “Efficient solutions for a university timetabling problem through integer programming”. In: *European Journal of Operational Research* 160.1, pp. 106–120.

Dekker, J. J., Banda, M. G. de la, Schutt, A., Stuckey, P. J., and Tack, G. (2018). “Solver-Independent Large Neighbourhood Search”. In: *CP 2018*. Ed. by J. N. Hooker. Vol. 11008. LNCS. Springer, pp. 81–98.

Dekker, J. J., Björdal, G., Carlsson, M., Flener, P., and Monette, J.-N. (2017). “Auto-tabling for subproblem presolving in MiniZinc”. In: *Constraints* 22.4, pp. 512–529.

Demeester, P., Bilgin, B., De Causmaecker, P., and Vanden Berghe, G. (2012). “A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice”. In: *Journal of Scheduling* 15.1, pp. 83–103.

Gecode Team (2019). *Gecode: A Generic Constraint Development Environment*. The Gecode solver and its MiniZinc backend are available at `https://www.gecode.org`.

Genc, B. and O’Sullivan, B. (2020). “A Two-Phase Constraint Programming Model for Examination Timetabling at University College Cork”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 724–742.

Güler, M. G., Geçici, E., Köroğlu, T., and Becit, E. (2021). “A Web-Based Decision Support System for Examination Timetabling”. In: *Expert Systems with Applications* 183, pp. 1–11. ISSN: 0957-4174.

Gurobi Optimization, LLC (2021). *Gurobi Optimizer Reference Manual*. URL: `https://www.gurobi.com`.

Hammersley, J. M. and Handscomb, D. C. (1964). *Monte Carlo methods*. London: Chapman and Hall.

Al-Hawari, F., Al-Ashi, M., Abawi, F., and Alouneh, S. (2020). “A practical three-phase ILP approach for solving the examination timetabling problem”. In: *International Transactions in Operational Research* 27.2, pp. 924–944.

June, T. L., Obit, J. H., Leau, Y.-B., and Bolongkikit, J. (2019). “Implementation of Constraint Programming and Simulated Annealing for Examination Timetabling Problem”. In: *Alfred R., Lim Y., Ibrahim A., Anthony P. (eds) Computational Science and Technology. Lecture Notes in Electrical Engineering*. Springer, pp. 175–184.

Kahar, M. M. and Kendall, G. (2010). “The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution”. In: *European journal of operational research* 207.2, pp. 557–565.

Kahar, M. M. and Kendall, G. (2015). “A great deluge algorithm for a real-world examination timetabling problem”. In: *Journal of the Operational Research Society* 66.1, pp. 116–133.

Keskin, M. E., Döyen, A., Akyer, H., and Güler, M. G. (2018). “Examination timetabling problem with scarce resources: a case study”. In: *European Journal of Industrial Engineering* 12.6, pp. 855–874.

Kristiansen, S., Sørensen, M., and Stidsen, T. R. (2015). “Integer programming for the generalized high school timetabling problem”. In: *Journal of Scheduling* 18.4, pp. 377–392.

Lach, G. and Lübbecke, M. E. (2008). “Optimal university course timetables and the partial transversal polytope”. In: *International Workshop on Experimental and Efficient Algorithms*. Springer, pp. 235–248.

Lach, G. and Lübbecke, M. E. (2012). “Curriculum based course timetabling: new solutions to Udine benchmark instances”. In: *Annals of Operations Research* 194.1, pp. 255–272.

Leite, N., Fernandes, C., Melício, F., and Rosa, A. (2018). “A cellular memetic algorithm for the examination timetabling problem”. In: *Computers and Operations Research* 94, pp. 118–138.

Leite, N., Melício, F., and Rosa, A. (2019). “A fast simulated annealing algorithm for the examination timetabling problem”. In: *Expert Systems with Applications* 122, pp. 137–151.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). “Optimal Speedup of Las Vegas Algorithms”. In: *Information Processing Letters* 47.4, pp. 173–180.

Maniezzo, V., Boschetti, M. A., and Stützle, T. (2021). *Matheuristics: Algorithms and Implementations*. Springer Nature.

McCollum, B., McMullan, P., Burke, E. K., Parkes, A. J., and Qu, R. (Sept. 2007). *The Second International Timetabling Competition: Examination Timetabling Track*. Tech. rep. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17. Queen’s University, Belfast (UK).

Muklason, A., Parkes, A. J., Özcan, E., McCollum, B., and McMullan, P. (2017). “Fairness in examination timetabling: Student preferences and extended formulations”. In: *Applied Soft Computing* 55, pp. 302–318.

Müller, T. (2016). “Real-life examination timetabling”. In: *Journal of Scheduling* 19.3, pp. 257–270.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). “MiniZinc: Towards a standard CP modelling language”. In: *CP 2007*. Ed. by C. Bessière. Vol. 4741. LNCS. The MiniZinc toolchain is available at `https://www.minizinc.org`. Springer, pp. 529–543.

Özcan, E. and Ersoy, E. (2005). “Final exam scheduler-FES”. In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 2. IEEE, pp. 1356–1363.

Parkes, A. J. and Özcan, E. (2010). “Properties of Yeditepe examination timetabling benchmark instances”. In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, pp. 531–534.

Qu, R., Burke, E. K., McCollum, B., Merlot, L., and Lee, S. (2009). “A Survey of Search Methodologies and Automated System Development for Examination Timetabling”. In: *Journal of Scheduling* 12.1, pp. 55–89.

Schaerf, A. (1999). “A Survey of Automated Timetabling”. In: *Artificial Intelligence Review* 13.2, pp. 87–127.

Shaw, P. (1998). “Using constraint programming and local search methods to solve vehicle routing problems”. In: *CP 1998*. Ed. by M. Maher and J.-F. Puget. Vol. 1520. LNCS. Springer, pp. 417–431.

Sørensen, M. and Dahms, F. H. (2014). “A two-stage decomposition of high school timetabling applied to cases in Denmark”. In: *Computers & Operations Research* 43, pp. 36–49.

Stuckey, P. J. et al. (2020). *The MiniZinc Handbook*. 2.5.5. `https://www.minizinc.org/`. Monash University.

Urli, T. (2013). “json2run: a tool for experiment design & analysis”. In: *CoRR* abs/1305.1112.

Woumans, G., De Boeck, L., Beliën, J., and Creemers, S. (2016). “A column generation approach for solving the examination-timetabling problem”. In: *European Journal of Operational Research* 253.1, pp. 178–194.

Al-Yakoob, S. M. and Sherali, H. D. (2015). “Mathematical models and algorithms for a high school timetabling problem”. In: *Computers & Operations Research* 61, pp. 56–68.

Al-Yakoob, S. M., Sherali, H. D., and Al-Jazzaf, M. (2010). “A mixed-integer mathematical modeling approach to exam timetabling”. In: *Computational Management Science* 7.1, p. 19.