PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Nara Torres Moreira

A MIP-Based Approach to Solve a Real-World School Timetabling Problem

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marcus Vinicius Soledade Poggi de Aragão

Rio de Janeiro
May 2015

Nara Torres Moreira

A MIP-Based Approach to Solve a Real-World School Timetabling Problem

Dissertation presented to the Programa de Pós–Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Mestre.

Rio de Janeiro, May 21st, 2015

Nara Torres Moreira

Nara Torres Moreira received her undergraduate degree in Computational Mathematics in 2011, from Universidade Federal de Minas Gerais (Belo Horizonte, Brazil). She also obtained a Master degree in 2015 at PUC-Rio in Computer Science focused on Combinatorial Optimization. During the master she held a scholarship from Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior (CAPES). She worked for Gapso company for over three years as Optimization Analyst and has been working for Accenture Digital as Data Science Analyst since beginning of 2015.

Bibliographic data

Moreira, Nara Torres

A MIP-Based Approach to Solve a Real-World School Timetabling Problem / Nara Torres Moreira; advisor: Marcus Vinicius Soledade Poggi de Aragão. — 2015.

136 f. : il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Geração de grades horárias para escolas. 3. Programação linear inteira mista. 4. Modelagem matemática. 5. Otimização combinatória. I. Poggi de Aragão, Marcus Vinicius Soledade. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, because I really couldn’t have asked for better.

Acknowledgement

I would like to express my gratitude to my adviser Prof. Marcus V. S. Poggi de Aragão, for the guidance throughout my master studies, for assisting me in understanding and discerning the relevant points in research, and for being available whenever I needed.

I gratefully acknowledge the Gapso company, where I was first introduced to real-world combinatorial optimization problems and where I could develop my technical skills, work and learn with excellent professionals, face challenges and improve my knowledge in the field of mathematical programming.

I am also grateful to all those who contribute to Trieda's development, especially to Francisco Fonseca and Marcelo Reis, from whom I have learned so much.

I place on record my gratitude to the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior (CAPES) for the financial support and to the Pontifícia Universidade Católica do Rio de Janeiro for giving me the opportunity to continue my studies.

I would like to thank to my parents, Adilson Assis Moreira and Milene Torres dos Santos Moreira, for supporting and encouraging me in my decisions and ambitions, for teaching me fundamental values of life, and foremost for always believing and trusting on me. Everything I have I own to them.

Finally, I wish to thank my friends, both from Rio de Janeiro and Belo Horizonte, for their patience, understanding and support during my master studies, and for the uncountable times they brought me fun. Particularly, I want to thank my friend and colleague Fabián Castilla, for so many useful advices, for helping me every time I need it and for making me laugh even on my most stressful days. Life is empty without friends.

Abstract

Moreira, Nara Torres; Poggi de Aragão, Marcus Vinicius Soledade (Advisor). **A MIP-Based Approach to Solve a Real-World School Timetabling Problem**. Rio de Janeiro, 2015. 136p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Timetabling problems look to schedule meetings in order to satisfy a set of demands, while respecting additional constraints. In a good solution the resulting timetables are acceptable to all people and resources involved. In school timetabling, a given number of lectures, involving students, teachers and classrooms, need to be scheduled over the week, while having to satisfy operational, institutional, pedagogical and personal restrictions. The difficulty of the problem has driven many researchers to work on solving approaches for it since the early 1960's. Finding an actual solution to a real world scenario implies satisfying many quality requirements and not ignoring the political issues, which turns the classical problem much more intricate. This work describes an approach based on mixed integer programming (MIP) developed for solving a real-world school timetabling problem and discusses ideas and issues faced during solution deployment phase for some Brazilian schools. In contrast to other works on school timetabling, teaching staff sharing between distinct school units are considered. Computational experiments were performed for scenarios whose number of school units varies from 2 to 15, number of teachers varies from 35 to 471 and number of classes varies from 16 to 295. Different strategies were combined aiming at converging to good solutions. Finally, results are evaluated and the best approaches are highlighted.

Keywords

School Timetabling; Mixed Integer Linear Programming; Mathematical Modeling; Combinatorial Optimization.

Resumo

Moreira, Nara Torres; Poggi de Aragão, Marcus Vinicius Soledade. **Uma abordagem baseada em programação inteira mista para resolver um problema do mundo real de geração de grades horárias escolares**. Rio de Janeiro, 2015. 136p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Problemas de geração de grades horárias visam agendar eventos a fim de satisfazer demandas, ao mesmo tempo que satisfazem restrições adicionais. Uma solução é boa se todas as grades horárias resultantes são aceitáveis para todas as pessoas e recursos envolvidos. Para a geração de grades horárias escolares, um número conhecido de aulas, envolvendo estudantes, professores e salas de aula, deve ser agendado ao longo da semana, enquanto limitações operacionais, institucionais, pedagógicas e pessoais devem ser satisfeitas. A alta dificuldade do problema tem levado muitos pesquisadores a trabalhar em abordagens de resolução para o mesmo desde o início dos anos 60. Encontrar uma solução aplicável em um cenário do mundo real implica em satisfazer vários requisitos de qualidade e em não ignorar questões políticas, o que torna o problema clássico muito mais intrincado. Este trabalho descreve uma abordagem baseada em programação inteira mista (MIP) desenvolvida para resolver um problema real de geração de grades horárias escolares e discute ideias e desafios encarados durante a fase de implantação da solução em algumas escolas brasileiras. Em contraste com outros trabalhos na área, o compartilhamento de professores entre diferentes unidades de uma escola é considerado. Experimentos computacionais foram realizados para cenários cujo número de unidades varia de 2 a 15, o número de professores de 35 a 471, e o número de turmas de 16 a 295. Diferentes estratégias foram combinadas, visando a convergência da procura por boas soluções. Por fim, os resultados são avaliados e as melhores abordagens são destacadas.

Palavras–chave

Geração de grades horárias para escolas; Programação linear inteira mista; Modelagem matemática; Otimização combinatória.

Contents

1 Introduction

This dissertation is based on experiences lived during development of the Brazilian commercial software TRIEDA. In this chapter, ideas are contextualized and a brief introduction, some references and main variants to the Timetabling Problem (TTP) are provided. General and important aspects of real world timetabling are also introduced, besides existing timetabling groups and conferences.

1.1 Introducing the basic timetabling problem

Timetabling problems are a kind of practical scheduling challenge faced by many organizations and important for their daily operations to succeed. The general problem can be described as the satisfaction of a set of demands through the scheduling of meetings between those who demand and the available resources, while respecting additional constraints. The resulting timetables should be feasible and acceptable to all people and resources involved.

Application areas vary widely from sport event timetabling to nurse scheduling, lectures scheduling and transportation timetabling. This work is about timetabling in the school environment, which is where the original problem was set.

In school timetabling, a given number of lectures, involving students, teachers and classrooms, need to be scheduled over a fixed period of time (typically a week), while usually having to satisfy a set of additional constraints of various types. For an actual and applied purpose, besides mandatory operational constraints, such as avoiding resources overbooking, it is essential to consider several solution quality constraints, which can be related to institution, pedagogical and personal needs.

Because schools differ a lot in their educational policies, both when considering the same country and especially more between distinct countries, the criteria for quality and even timetable feasibility definition also depends on each educational system. In this work, the Brazilian school educational system is embraced. Studies based on the German school system can be found at (Marte, 2002), on the Greek system at (Birbas et al., 1997), (Birbas et al., 2009) and (Valouxis et al., 2012), and on Brazilian system at (Santos et al., 2012). An overview of school timetabling research is provided by (Pillay, 2010), describing the most common operational and quality requirements and the different solving methods that have been used.

University timetabling problems have also been very studied and have many similarities with the school version. Studies based on a Turkish university system can be found at (Guenalay and Sahin, 2006) and on a Greek university system at (Daskalaki and Birbas, 2005). For more complete approaches, one can check at (Carter, 2001) for a Canada university system and at (Murray et al., 2007) for an American university system.

The problem is known to be difficult and therefore many researchers have shown an interest in it since the early 1960's. Schaerf (Schaerf, 1999) showed for a classical version of the school timetabling problem that it is NP-complete. Furthermore, enhancing a model with real-world practical and quality requirements turns the problem much more intricate and can increase even more its complexity. For instance, considering the compactness requirement for students timetables (detailed in section 2.2.2), (Asratian and Werra, 2002) shows that deciding whether a timetable that guarantees compactness for the students exits is NP-complete by itself.

1.2 Brazilian school timetabling problem

In Brazil, basic educational system is split in 3 stages:

1. Fundamental Education I
2. Fundamental Education II
3. Upper secondary education

Fundamental Education is mandatory for children ages $6-14$. There are 9 "years" (as opposed to the former 8 "grades"). The current "First Year" broadly corresponds to the former Pre-School last year of private institutions, and its aim is to achieve literacy. Generally speaking, the only prerequisite for enrolling in first year is that a child should be 6 years old.

The Federal Council of Education establishes a core curriculum consisting of Portuguese language, history, geography, science, mathematics, arts and physical education (for years 2, 3, 4 and 5). As for years 6, 7, 8 and 9, one or two foreign languages are also compulsory (usually English and an optional language).

Each education system supplements this core curriculum with a diversified curriculum defined by the needs of the region and the abilities of individual students.

Fundamental Education is divided in two stages and takes 9 years long. During the first stage, each group of students is usually assisted by a single teacher. At the second stage, there are as many teachers as subjects.

Students must have finished their Fundamental education before they are allowed to enroll in Secondary education ("Ensino Médio"), which takes 3 years. Secondary education core curriculum comprises Portuguese (including Portuguese language, Brazilian and Portuguese literature), foreign language (usually English, also Spanish and very rarely French), History, Geography, Mathematics, Physics, Chemistry and Biology. Philosophy and Sociology, which were banned during the military dictatorship (1964 to 1985), became compulsory again. The courses provided during this period are essentially designed to allow a young person to enter an university. A minority of schools provides professional training along with mainstream secondary education. Professional training courses usually last 2 years and can be taken during the 2nd and 3rd years of Secondary education.

Once a student has successfully completed secondary education, he may continue his studies at a public or private university. To enter a public university, students must sit an entrance exam, known as "vestibular". Entrance exams to a private university are often little more than a formality and, as a consequence, public university degrees are "usually" valued much more than those from private institutions.

The normal practice in Brazilian schools, both public and private, is to mix students of all study directions together in the same class. Due to the entrance exam to universities, it is common though at private schools for students at last year of secondary education to have extra specialized classes, which depend on their choice for undergraduation course. Data from (Wikipedia).

1.3 Practical and realistic formulation

Both school and university timetabling problems have been studied for over 50 years now. Many potentially useful solving algorithms and techniques have been discussed and offered, but unfortunately much of the work in this area has been conducted using artificial data sets or based on greatly simplified versions of actual problems. Often the considered scenario has already preassigned sets of entities, like professors to classes in the case of school timetabling or students to course sections in the case of university. Besides, the methods developed have most of the time been restricted to a single university department or school unit, instead of being extended to the solution of actual university and school problems of any large scale. Whenever the different departments or units have dependencies, like sharing resources, usually teaching staff, such partition of the problem is relevant and can therefore be prejudicial to the final solution quality and acceptance.

For the university environment, we draw attention to (Carter, 2001) and (Murray et al., 2007) as those who made efforts on solving very realistic versions of the problem for universities in Canada and in USA. They described both the methods used for solving the problems and the challenges faced during the process of deploying solutions. Also, earliest implementations of a decision support system for academic scheduling can be backtrack to 1986, at (Kassicieh et al., 1986), for University of New Mexico in USA.

The major differences between many of the problems studied and their real life counterparts are the additional complexity imposed by course structures, the variety of quality constraints imposed, and the distributed responsibility for information needed to solve such problems at a wider level. Besides, as pointed out in (Murray et al., 2007) for university course timetabling scenario, the complexity of solving actual timetabling problems can increase considerably beyond that represented in standard formulations of the problem. As the complexity increases, it is easy to be caught in the dual bind that the problem is both more challenging to develop an effective solution approach for, and this approach is less likely to be usable on other educational institutions. Every institution has its own particularities, which make it harder to build a single and general solver.

As observed by Carter (Carter, 2001), there are still few published papers that described actual implementations of course timetabling. From what we know, the first example of an automated and integrated course timetabling and student scheduling system across a institution was developed for the University of Waterloo between 1979 and 1985. Although the system was implemented 29 years ago, much of the discussion is still very relevant. By 2001, Carter expressed his belief that there was still no system for solving the large-scale course timetabling problem with such level of mathematical sophistication.

By 2006, Guenalay and Sahin introduced at (Guenalay and Sahin, 2006) a Decision Support System (DSS) for the university timetabling that allows the direct involvement of the decision maker. They affirmed that their model was the first one which simultaneously combined such a DSS approach with a goal programming optimization tool.

At (Benli and Botsali, 2004) is documented a DSS for scheduling courses at Bilkent University. At (LLC, 2007 - 2014) the Unitime software for university scheduling is described and widely documented. It is an open source system used at Purdue University and specially developed for it, and also the focus of (Murray et al., 2007).

1.4 Main similar formulations and their differences

For solving the high school timetabling problem, this work used an integer programming formulation together with strategies for helping the convergence of the searching process. This work is based on the development of the educational timetabling software TRIEDA and, because it is a commercial software, it has been modeled to be as portable and flexible as possible. Particularly, the system embraces most common rules and cases found in Brazilian universities and schools.

Among the existing published works that considered more realistic formulations for high school timetabling problems, we highlight (Birbas et al., 2009) and (Valouxis et al., 2012) as those whose embraced problems have more similarities with the one considered in this work. Several practical issues emphasized by them were also faced during TRIEDA's development and are referenced along this work. Still, there are many differences between their timetabling problems and the one here presented.

The most significant and *unanimous* difference is that other school timetabling works do not consider multiple and integrated school units. We found it common in Brazil that large schools with several units spread through a city share teaching staff. When scheduling lessons, such resource sharing implies both that we can not solve school units independently and that professor daily displacement between units must be considered, which increases size and complexity of the problem. No record of studies in literature that handle professors displacements between multiple integrated units with equal in-depth details was found.

Next relevant difference is related to available times of professors. The more constrained are professors availabilities, the more realistic, but also more difficult, is the resulting problem. The related limitations of studies found in literature are to consider professors availabilities only as preferences instead of considering them strictly; to consider all or most professors as full-time ((Birbas et al., 2009)); to consider professor availability only in a day level instead of detailing availability per time period ((Birbas et al., 2009), (Valouxis et al., 2012)); or simply to bound professors workload ((Birbas et al., 1997)).

Further recurrent difference is related to assignment of classrooms. It is true that usually in school timetabling problems each class already has a preassigned classroom. This was not different in scenarios experimented by this work, but still there are some exceptions. Some courses, usually practical subjects, require special rooms, like laboratories or sport courts, which are not unique per class. Many studies do not consider rooms assignments ((Birbas et al., 1997), (Birbas et al., 2009), and (Santos et al., 2012)). With the aim of being as flexible and generic as possible, this work does include rooms assignments.

In terms of solving method, both works (Birbas et al., 2009) and (Valouxis et al., 2012) used a hybrid approach, decomposing the problem in phases and using integer programming for solving sub-problems. That is a quite reasonable and common strategy, where one replaces the solving of a hard and large problem with the solving of some easier and smaller problems. The problem is that such decompositions cause reduction of the solution search-space and there is usually no guarantee that better or optimal solutions are not eliminated during such reduction. This present work attempts to solve the whole problem using an integer programming approach, but avoiding this kind of decomposition and therefore guaranteeing that the optimal solution is never thrown out.

Further differences and similarities between other educational timetabling problems and the one here presented can be identified along this work.

1.5
Data accuracy and Politics

For large educational institutions it is common that responsibility for constructing the timetables is distributed among the different academic units. Each unit has more control and an intimate knowledge about its own teaching staff, physical facilities, courses and demands. Besides, many times there is information that is not stored in any database — practical unofficial knowledge, that can only be obtained by contacting departmental timetablers. On the other hand, there are often sharing of resources and demands between units, which connects their timetables and precludes them to be independent from each other. Murray, Müller and Rudová observe at (Murray et al., 2007) that maintaining each department's sense of ownership in the timetables that are produced is an important factor in their acceptance of the solutions produced by an automated timetabling process, and that the process needs to be one that assists them rather than replaces them. Achieving this corresponds basically to have accurate input data and to understand all aspects that characterize a solution as a bad or a good one.

Obtaining coherent and correct data is a particular important and sensitive issue. Because data accuracy depends on each department, it is extremely important the data collection phase of an automating process to have a deep involvement of all those who usually operate in the traditional manual process. Any inaccuracy of data can result in a bad or even non-deployable solution.

Murray, Müller and Rudová tells at (Murray et al., 2007) that inequity in the quality of time and room assignments received by different departments and faculty members doomed a previous attempt at automating the timetabling process at Purdue University. A similar situation was faced by TRIEDA in July of 2014 while an attempt of deploying a solution in a Brazilian university.

Automating people assignments is different from automating other processes — the optimization factor is definitely **not** more important than people satisfaction. Only keeping this in mind it is possible to obtain an actual deployable solution.

As wisely pointed out in (Carter, 2001), practical course timetabling is 10% graph theory, and 90% politics. Carter tells that when they first began designing a timetabling system for University of Waterloo, they were warned that they cannot dictate when professors will teach courses. Consequently, they were told that course timetabling could not work or, more precisely, that they cannot assume that the timetable can use a clear slate. This is because part-time professors may be available only on certain and specific days or times. University professors often have other commitments and industrial research projects. Teaching is only a part of their job, and probably less than half of their time is devoted to teaching. Similar statement is valid for high school teachers. It is very common that they teach in more than one school or have other activities. It follows that it is essential to allow professors to restrict their available timetables as much as they want.

Guenalay and Sahin describe in (Guenalay and Sahin, 2006) an approach for university timetabling with a goal programming optimization tool in order to process the instructor teaching time preferences. As they observe, using a multi-objective function for considering instructor available preferences may not be enough for ensure a reliable and acceptable solution.

Professors' availabilities and preferences issue should therefore not be underestimated.

The primary design goal is not necessarily finding a true optimal solution, but to assist academic timetablers with the problem of building a good and better timetable in an efficient way.

1.6 Timetabling Groups and Conferences

Many groups, conferences, competitions and companies dedicated to solving timetabling problems have grown in past years. Following some of them are listed.

– Practice and Theory of Automated Timetabling Conferences (Patat)

PATAT

The International Series of Conferences on the Practice and Theory of Automated Timetabling (Pratice and Theory on Automated Timetabling) is held biennially since 1995 as a forum for both researchers and practitioners of timetabling to exchange ideas. An important aim of the Conference is to align the needs of practitioners and the objectives of researchers, which is achieved through the presentation and application of leading edge research techniques.

– International Timetabling Competition (ITC)

international
TIMETABLING COMPETITION

The International Timetabling Competition (ITC) has happened in 3 editions, in 2002, 2007 and 2011. Not much information was found about the first ITC, in 2002. The second edition (International Timetabling Competition (ITC), 2007) focused on examination and courses timetabling problems and was sponsored by PATAT and WATT. The third edition (International Timetabling Competition (ITC), 2011) was devoted to High School Timetabling and sponsored by PATAT, EventMAP and CTIT.

The competition is composed of three separate rounds, which represent distinct problems within the area of educational timetabling both from a research and practical perspective. It aims especially to encourage the alignment of research with practice by offering real-world instances of timetabling problems for solution. Although not all aspects of the real world problem are included, more depth and complexity have been introduced significantly at each new edition.

– The Curriculum-based Course Timetabling

Curriculum-Based Course TimeTabling

Following the spirit of the ITC, the website (Curriculum-Based Course Timetabling Project) provides a portfolio of formulations for a version of the Curriculum-Based Course Timetabling problem. It is possible to download instances and solutions contributed from the research community; validate your solutions online or download the source code of a solution validator; insert solutions, lower bounds, instances, and links of interest for the community; and generate random instances. All problem instances available to download are based on real data from various universities.

– Research Group on Scheduling and Timetabling in Udine

research group on Scheduling and Timetabling

The Scheduling and Timetabling research group in Udine is working on the solution of scheduling problems using both local search techniques and their hybridization with other paradigms. The researchers involved in the group have developed both academic prototypes and production systems for the solution of practical scheduling problems, such as workforce scheduling, shift design, timetabling, and sport scheduling. More information at (Udine).

– EURO Working Group on Automated Timetabling (WATT)

EURO The Association of European Operational Research Societies

WATT EURO Working Group on Automated Timetabling

WATT is the EURO Working Group on Automated Timetabling, formed to discuss, promote, and perform research into automated timetabling problems and methods. The WATT community was launched in the wake of the first international conference on the PATAT in 1995. Different timetabling categories are studied and for each one datasets and references to literature are provided. Further information at ((Watt)).

– Multidisciplinary International Scheduling Conference (Mista)

Mista is a conference series held biennially since 2003, which serves as a forum for an international community of researchers, practitioners

MISTA
Multidisciplinary International Scheduling Conference: Theory & Applications

and vendors on all aspects of multi-disciplinary scheduling. The aim is to bring together scheduling researchers and practitioners from all the disciplines that engage with scheduling research. Further information at (Mista).

– Companies and Software

  Some examples of companies and software specially dedicated to solving timetabling problems are (Mimosa Scheduling Software, 2014), (Event MAP, 2014), (LLC, 2007 - 2014) and (Asc Timetables). This present work is itself based on the Brazilian software Trieda (Trieda).

1.7 Dissertation Outline

This work is organized as follows:

– Chapter 2 introduces the complete timetabling problem considered in this work.

– Chapter 3 shows the mathematical formulation developed for the problem.

– Chapter 4 presents different approaches and strategies for solving the MIP formulation.

– Chapter 5 introduces a set of distinct scenarios used in this work and displays a large range of computational experiments performed for the different scenarios.

– Chapter 6 contains the conclusions of this work.

– Appendix A presents XHSTT format and an attempt to convert a XHSTT problem into a Trieda's problem.

– Appendix B presents complete computational results.

2 Trieda's Timetabling Problem

School timetabling problems vary a lot from country to country. This chapter provides a complete definition to the timetabling problem for Brazilian elementary education system considered in this dissertation.

2.1 Introducing the system

TRIEDA is an academic planning system for educational institutions, designed as a multi-user application with a completely web-based interface. Unlike many academic planning systems and timetabling researches, which consider simplifications of the actual problem, TRIEDA makes all necessary decisions so that a complete, optimized and applicable solution is provided.

Timetabling problems vary a lot depending on the kind of educational institution. The system has nowadays two distinct modules: one for solving high school timetabling problems and another for solving university timetabling problems. For each one, TRIEDA uses a completely different solver. This dissertation focus only on the high school module.

The system is based on a "demand-driven" philosophy where students first choose their courses, and having knowledge of the complete institution structure and available resources, the aim is to provide a complete and feasible solution that maximizes the number of satisfied requests while respecting a set of didactic-pedagogical requirements. For the university module, reducing costs is also strongly aimed. For the school module, some didactic-pedagogical requirements together with professors satisfaction are usually the most important issues.

2.1.1 Simulation usability

Carter tells us in (Carter, 2001) that, in the Fall of 1988, the University of Waterloo opened the Davis Building. A total of 40 classrooms across campus were replaced by 30 generally larger classrooms in the new building. The number of rooms decreased, but the total number of seats increased. The

Scheduling Office was very concerned about how the new space would impact space allocation. A timetabling automated system developed for the university effectively was used for simulating the new scenario and solved a potentially serious planning problem.

Therefore, as it was exemplified, the timetabling system can also be used as a fast "What if?" tool to evaluate proposed changes in resources or institution rules.

2.1.2 Multiple scenarios

Through the system interface the user can easily register each entity and resource of the institution and its quality requests. The set of all registered information for the problem defines a scenario. The system allows each user to create and manage multiple scenarios, which is very useful when comparing and evaluating the impact of different situations.

2.1.3 Interface and solver interaction

Interaction between the user interface and the solver is made by XML files. Once all problem data is registered by the user such that one has a consistent scenario, an optimization request can be made. Such request generates a XML input file and calls the solver, which reads the input file, loads data and starts the optimization process. Once the final solution is obtained, a XML output file is written and the solver process is ended. Following, the application loads the solution and user interface shows all results.

2.1.4 Manual changes

A very common requirement in logistic and planning systems is to allow the user to make manual changes in a solution. TRIEDA allows user to manually create new classes or modify existing classes, as long the result does not violate basic and necessary operational constraints.

2.1.5 Initial solution

Another very common situation is that small modifications take place in problem data, resulting in a slightly different scenario. Most frequent changes in input data are related to professors availability, which have been shown to be quite fluctuating. Supposing one has a solution for an initial scenario, and some small and local changes take place in data, it seems senseless or a waste to generate a new solution from scratch, specially if scenario is big and solving it takes a long time. Bearing this in mind, a really useful functionality is to provide the solver with an initial feasible solution and the portion of such initial solution that should be fixed.

2.1.6 Reasons for non–satisfaction of demands

As discussed in section 1.5, data quality is essential for obtaining a good, feasible and applicable solution. Unfortunately, data inaccuracy is though very frequent and therefore the system should be prepared to deal with it, as long it is possible.

Besides the possibility of data inaccuracy, it can be hard to affirm if the demand of a scenario can be fully satisfied, specially when it is a large instance. Analyzing both input data and solution for a real world problem is a hard task. As an user auxiliary functionality for solution analysis, the system exhibits for each non–satisfied demand possible explanations for the non-allocation, which can help both to identify input data errors and to understand resources limitations.

2.1.7 Virtual professor tips

Because there is in general no insurance that there are enough teaching resources for fully satisfying the demand, the concept of virtual teaching resource is used (see at 2.2.1). Virtual resources should though be used as little as possible.

An extra functionality of the system for helping the user while managing a solution is called “virtual professors tips”. For each virtual professor assigned to a solution class, it may be exhibited a tip or a suggestion for a change in some professor availability that would lead to a possible substitution of that virtual resource for a real professor. This can be quite useful for scenarios where professors availability is very limited, but it is ineffective when professors are full-time.

2.2 Introducing the school timetabling problem

Unlike university problems, high school's students have low or no freedom at all in their choices for courses. Usually students are previously clustered into sections, in the sense that if two students are in the same section, then they have the same demands and should attend to the same lessons. Therefore, in a school timetabling problem the number of sections per course is known and assignments are made for each pre–defined cluster of students instead of for each single student.

Besides, in high school, sections usually have pre–assigned rooms. This is not an obligation though. For practical courses, it is common that several sections attend to lessons at the same laboratory. For sport classes, it is even common that two or more classes simultaneously share a physical space, like a sports court. But indeed, only in a minority of cases shared rooms are involved.

A solution is composed of a set of courses sections that should be offered in order that demands are satisfied. For each course section it must be decided:

– the classroom where section's lessons take place. Usually each section has a single possible room, but for practical courses it is to be decided;
– the time slots of each lesson;
– the professor assigned to the section;

Incomplete solutions (course section lacking classroom or scheduled time, for example) are not acceptable.

The planning horizon, or *teaching period*, is a week. It refers to the timetable duration, so that it is considered that classes don't change from a week to another in the same year or planning term.

2.2.1 Entities and Concepts

Time slot Each time slot has a starting time, an ending time and a weekday. For instance, Monday from 14:00 to 14:50. Besides, time slot duration is the number of minutes between the starting and ending time of a time slot.

Shift A shift is a label for a set of slot times.

It is mandatory that every time slot of a timetable belongs to some shift; a timetable can have time slots of different shifts; and a shift can have time slots of distinct timetables.

There is no constraint for associating a time slot and a shift, which means that a shift can be seen simply as a label (usually called "Morning", "Afternoon", "Evening", "Integral", "Morning-Afternoon", etc) and time slots of the same shift can overlap each other or have distinct durations.

The main role of the concept of shift is to define in which time slots a student can have classes, as discussed ahead at “Student schedule” in section 2.2.2.

Calendar Timetable Corresponds to the way a school term week is split in the sense of shifts and time slots.

Table 2.1 shows an example of timetable with 2 shifts (morning and afternoon), each one with 4 class-times and a 30-minutes break between the 2 first class-times and the 2 last class-times of the shift. All class-times have 50 minutes duration. From Monday to Friday all time slots are available, on Saturday only the morning time slots are available and on Sunday no time slot is available (unavailability is represented by a dash).

Shifts	Time Slots	Mon	Tue	Wed	Thu	Fri	Sat	Sun
MORNING	08:00 to 08:50							-
MORNING	08:50 to 09:40							-
MORNING	10:10 to 11:00							-
MORNING	11:00 to 11:50							-
AFTERNOON	14:00 to 14:50						-	-
AFTERNOON	14:50 to 15:40						-	-
AFTERNOON	16:10 to 17:00						-	-
AFTERNOON	17:00 to 17:50						-	-

Table 2.1: Morning-Afternoon Timetable.

It is possible to register in the system as many timetables as necessary. This way, besides the timetable at 2.1 we can *also* have the one at 2.2.

This timetable has also time slots in 2 shifts, now afternoon and evening, but here class-times have 60 minutes duration. It has 3 class-times in the Afternoon shift and 4 class-times in the Evening shift, the break-times are completely different from break-times of Morning-Afternoon Timetable, and there is no time slot available at weekends (Saturday and Sunday).

Shifts	Time Slots	Mon	Tue	Wed	Thu	Fri	Sat	Sun
AFTERNOON	13:00 to 14:00						-	-
AFTERNOON	14:00 to 15:00						-	-
AFTERNOON	15:00 to 16:00						-	-
EVENING	18:00 to 19:00						-	-
EVENING	19:00 to 20:00						-	-
EVENING	20:20 to 21:20						-	-
EVENING	21:20 to 22:20						-	-

Table 2.2: Afternoon-Evening Timetable.

Time slots of the same timetable must have the same duration and can not overlap each other. Time slots of different timetables are independent: they can be completely different and even overlap, as the examples have shown.

Session of a day Sessions of the day are the commonly known concept for Morning, Afternoon or Night. The concept of sessions of day is necessary only to assist in obtaining compact timetables for professors.

Course A *course* is a "chair" or "subject" that a student can take. Common examples in high-school are "Mathematics", "Geography" or "Biology".

Every course has a total number of credits per week, with all the credits necessarily having the same duration. This means that if the student John Mayer takes classes of a course with N credits of 50 minutes duration, exactly N time slots of 50 minutes of John's week schedule must be assigned to lessons of this course.

Every course has a set of time slots to which it can be assigned. These time slots can belong to different timetables, as long they have the same duration.

Every course can be assigned to a specific set of classrooms where it can be held. If there is no specific classroom associations, it is assumed that the course can be held in any classroom.

Curriculum A *curriculum* is the set of all courses taken by a student during a specific school studying year.

Campus, Block and Classroom The basic physical structure of a school consists of:

– *Classroom*: room where classes can be held.

- *Block*: a block represents a school unit. Every block has a set of classrooms, between which displacement time is insignificant and is thus considered zero, a set of professors available for teaching in that block, and a set of students demands of that block. Usually each student has classes in only one block, but this is not a rule. Distinct blocks can share teaching staff resources, which makes it necessary to limit some assignments depending on the time necessary for the professor to move between blocks.
- *Campus*: a campus is the higher-level structure of the problem and gathers all resources and demands information, such as blocks, professors and students requests.

Offer An *offer* is made of a curriculum, shift and campus. Its interpretation is that courses of this curriculum are offered at the campus at the specific shift.

Student–Demand A student–demand is an individual request of a student for a course.

Class A class refers to the same group of students that will be taught a set of courses. Every time we refer to a student along this dissertation, we actually mean a class. This is because Trieda was originally designed to handle individual students requests in university environments, but evolved later to work with high school environments. Since in high schools students are usually already grouped, which means that students of a group have the same needs and requests, in the school module a class is represented by a single student and either classrooms have capacity equal to 1 or courses have a maximum number of students limited to 1, except for situations where classes are expected to be merged.

Lesson A lesson refers to a particular course being taught to a class by a teacher at some consecutive periods of a day.

Course Section A *section of a course* is a class (set of students) and its associated set of lessons for the course.

In a complete and feasible solution, each course section has the following attributes:

- a set of students (class);
- a classroom where the lessons take place;

– time slots for each course lesson, totaling the number of credits of the course;

– a professor who teaches the lessons.

Professors Each professor is a teaching staff resource that can be assigned to classes, according to its features. Every professor has:

– its own available timetable,

– a list of courses that he/she is capable of teaching,

– a priority number, 1 or 2, that indicates how important is that professor for the institution.

Virtual Professors The most basic solution restrictions are based on resources availabilities. There is no assurance in general there are enough professors to meet the whole demand. Therefore, the concept of *virtual professor* is introduced.

Each *virtual professor* means the hiring of a new professor by the institution. Whenever there is no feasible solution with full satisfied demand due to absence of professors, and only in this situation, virtual professors should be created by the solver and used until demand is fulfilled.

Since a virtual professor is just a prediction of a new profile, it has default settings, full-available timetable and is not considered at some professor quality restrictions.

The need of virtual teaching resources is also due to the usual floating feature of professors availability.

2.2.2 Constraints

Constraints can be either hard or soft. A hard constraint is inviolable and a soft constraint has its violation minimized by the objective function.

Following all decision-making rules considered by this work are introduced. These rules are organized as being mandatory or optional constraints. An optional constraint can be either hard or soft, while the only soft mandatory constraint is demand fulfillment, i.e., all the others are hard.

Mandatory Constraints

Demand fulfillment The first objective of the problem is maximizing demand satisfaction. Since there is no guarantee that the whole demand can be assigned, this is not considered as a hard constraint, but the first goal in objective function. Therefore, a way of controlling the satisfied demand is necessary.

Classroom Capacity Each classroom has a mandatory register called ”room capacity” which indicates the maximum number of students that it can simultaneously hold. Due to this physical capacity, no class can have more students than the capacity of the room where the lessons take place.

Same classroom for each course section For each course section, all the credits of the course must be taught at the same classroom.

Possible classrooms for each course Each course has a set of classrooms where its lessons can be taught. This set must be respected. The only exception is if it is empty — then lessons can be assigned to any available classroom.

One professor for each course section For each course section, all the credits of the course must be taught by the same professor.

Capability for teaching courses For a professor to be assigned to classes of a course it is necessary that the professor is capable of teaching that course. So, each professor must have a register of courses that he is capable of teaching and for every solution the professor assignment must respect these registers.

Time Availability Every resource, i.e. professors and classrooms, has its own timetable availability that informs the time slots to which it can be assigned. Courses have also their own timetable availability.

For instance, consider the table 2.3 with the availability for professor Paul McCartney.

Shifts	Time Slots	Mon	Tue	Wed	Thu	Fri	Sat	Sun
MORNING	08:00 to 08:50		-				-	-
MORNING	08:50 to 09:40						-	-
AFTERNOON	16:10 to 17:00				-		-	-
AFTERNOON	17:00 to 17:50						-	-

Table 2.3: Available time slots.

Professor McCartney can**not** be assigned to classes at any time slot on Saturdays or Sundays, neither from 8:00 to 8:50 on Tuesdays nor 16:10 to 17:00 on Thursdays. In the others time slots he is available.

Student schedule Every resource has its own available timetable and it defines at which time slots they can be assigned. Students do not have an explicit timetable of availability, however there is still the need of constraining their possible time slots.

Since it is possible that many offers exist at different shifts, it makes sense to limit the student's possible time slots to those which belong to the student's shift. Each offer specifies a major's curriculum, a shift and a campus where it is being offered, and every student is usually associated with one offer.

For example, Etta James is a student of Computer Science, curriculum version *v–98*, offered at the Morning shift and at campus I. It follows that Etta James can only be assigned to classes at time slots present at the Morning shift. Besides that, Ray Charles is a student of Economic Science at Morning-Afternoon shift at the same campus and both students need to take classes of Calculus I. They could be assigned to the same Calculus I section, as long its lessons are held at time slots belonging to both shifts, i.e., in the Morning.

No time slots overlapping The most basic rule for any timetabling problem is the no overlapping in resources and students timetables. In other words, any pair of classes assigned to the same entity (classroom, professor or student) at the same day and same time is forbidden.

Compact students timetables A "gap" or a "hole" in a student's timetable is an idle time window between classes *at the same day*.

Suppose the student Peggy Lee attends to a class from 8:00 to 10:00 on Monday, and then from 11:00 to 12:00 on the same day. The idle time window between 10:00 and 11:00 is an "1-hour gap" in the morning.

Breaking times between time slots of a calender should be ignored. For example, if Peggy attends on Tuesday to a class from 08:00 to 08:50, a break from 08:50 to 09:10, and other classes from 09:10 to 12:00, there is no gap due to the 20-minutes break. Also, if her classes resumes at 13:00, and the time window 12:00–13:00 is an interval at her shift (lunch time, probably), there is no gap due to the 60-minutes break.

Time for professor displacement between blocks Whenever a professor is assigned to any pair of classes at the same day, which take place at different blocks, the minimum traverse time spent between both spots must be considered.

Number of credits of a course Each course has a total number of credits. Every student attending to a course must be assigned to all its credits.

Maximum class size Each course has a value that indicates the maximum number of students that is allowed in the course lessons. Therefore, every course section must respect such maximum class size.

Controlling Virtual Professors Usage Minimizing the number of credits assigned to virtual professors is considered the second most important aim in the multi-objective function of timetabling problems, second only to maximizing satisfied demands. The solver should use a virtual professor if and only if there is not enough teaching staff resources available.

Optional Constraints

The following constraints are optional, which means that with their absence a consistent solution is still produced. Unlike the mandatory constraints, there are parameters to control if they should be considered or not. In case an optional constraint is considered, there is still the possibility of making it a hard or soft constraint. This varies according to the constraint type, as explained in more detail bellow.

Credits Split Rules Courses may require more than 1 weekday to have all their credits scheduled. Since not every credits split is appropriate, the concept of credits split rule is introduced to define suitable splits.

For example, consider the following credits split rules at table 2.4.

Course(s)	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
4-credits Courses	2		2				
6-credits Courses	2		2	2			
6-credits Courses	3		3				
INF332 (4 credits)	2	2					

Table 2.4: Course credits split rule.

The first 3 rules are said to be general, while the fourth is a specific rule. The first rule says that 4-credits courses should be split into lessons of 2 credits each along 2 weekdays preferably non-consecutive. Similarly, 6-credits courses should be split into lessons of 2 credits each along 3 weekdays; or into lessons of 3 credits each along 2 weekdays. At last, the course $INF332$ should be split into 2 days, preferably consecutive, and each day with 2-credits lessons.

If there is a course with m credits but no m-credits split rule, then m-credits course split is free. Credits split rules are optional, but when present they are hard constraints.

Whenever there is a specific credit split rule for a course, it must be respected, even if other generic split rules exist for the same number of credits.

Although the system does not make it mandatory for a course to have associated credits split rules, in practice this is an important issue and widely applied.

Single lesson for each course section per day It may be undesirable that several non-consecutive lessons of the same course section are assigned to the same day. In that case, it is possible to ensure that every course section has at most one lesson (set of consecutive periods) per day.

Maximum number of weekdays that a professor is available It is common that professors teach in more than one institution or have other activities. For this reason, regardless the available timetable of professors, it is possible that they have a limited number of days for teaching at the institution. For instance, although professor McCartney is available from Monday to Friday (see 2.3), it is possible that he can take only 3 days of week, to be chosen between Monday to Friday, for teaching at the institution. Then, each professor has an integer attribute that indicates the maximum number of weekdays that he can be assigned to, with default value equal to 7.

Minimum number of credits at the day for a professor to teach Regardless the available timetable of professors, it is possible that they request a minimum number of credits for teaching at a day. If professor Eric Clapton requires a minimum of 3 credits, then he is assigned to classes on Monday (or any other day) only if the total of credits on Monday is equal or greater than 3. The default minimum value is 1 credit, i.e., the trivial case.

Minimizing the number of busy sessions and days in professor's timetable The more compact is a professor timetable, the better. Thus, another possible aim of the solver is to *minimize* the number of busy sessions and busy days in professor's timetable, instead of just establishing a hard minimum number of credits per day and a hard maximum number of days in the week.

Daily rest period For professors assignments to be applicable, some labor laws should be respected. The rest period law says that a minimum rest period between 2 labor days is needed. In Brazil this minimum rest period is 11 hours, which means, for instance, that if professor Tracy Chapman teaches until 22:00 of Monday, she can not be assigned to classes earlier than 9:00 on Tuesday morning. Whenever they are considered, minimum rest periods are hard constraints.

Minimum and maximum professor workload Among the labor laws which must be respected, so that professors assignments are applicable, there is the minimum and maximum professor workload. It may be forbidden that a professor has his workload reduced more than $k\%$ of his previous semester workload at the institution. This constraint guarantees some stability to employees. It may also be forbidden for a professor to be overloaded.

For instance, professor Steven Tyler can have no more than 10% of his previous semester workload reduced and has a weekly maximum workload equal to 24 credits.

Number of professor displacements between blocks in a day Whenever a professor is assigned to any pair of classes at the same day and taking place at different blocks, a maximum of 1 displacement can be established.

For instance, if Professor Marvin Gaye teaches at block A on Monday morning and at block B on Monday afternoon, then he has already 1 displacement on Monday to go from A to B. He cannot be assigned to any block on Monday other than B after he moved to B for the first time in the day, and likewise any block other than A before he had moved to B for the first time in the day.

Clearly, this restriction implies that a professor is never assigned to more than 2 blocks at the same day, but it is stronger than that, since a sequence of blocks $A \rightarrow B \rightarrow A$ in a day involves 2 blocks (ok) and 2 displacements (not ok).

Maximum number of blocks assigned to a professor in a day A weaker constraint than the previous one is the limitation on the number of distinct blocks where a professor has classes to teach at the same day. The appropriate limit value varies with the physical distribution of the blocks. The more distant the blocks are from each other, the lower is the limit, since longer travel time is necessary.

Compactness in professor's timetable A "gap" or "hole" in a professor's timetable is an idle time window between classes at the same *session of a day*.

Suppose professor Meschiya Lake teaches from 8:00 to 10:00 on Monday, and then from 11:00 to 12:00 on the same day. The idle time window between 10:00 and 11:00 is an "1-hour gap" in the morning.

Breaking times between time slots of a calender should be ignored. For example, if a professor is assigned to a class from 08:00 to 08:50, a break from 08:50 to 09:10, and another class from 09:10 to 10:00, there is no gap due to the 20-minutes break.

Gaps are undesirable. They lower professor's satisfaction and can sometimes even increase the institution cost, because according to labor laws the institution might have to pay the professor for the idle time. Compactness requirements for professors are particularly important when teachers may work in multiple institutions.

Professor preference for teaching courses Every professor has a list of courses he is able to teach. Among these courses, though, it is possible that there is a preference of the professor for teaching one instead of another. For representing this preference, every such pair [professor, course] has an associated integer value that ranges from 1 to 10 and indicates the professor's preference for teaching that course, where 1 is the highest possible preference and 10 is the lowest.

The importance given to professors preferences depends on the institution, i.e., how relevant is this aspect to it. Usually, professors' preferences have a low weight when compared to others requirements.

2.3 Data quality

Here we strengthen the importance of input data quality, an issue already introduced in section 1.5. Following the most common problems found in input data are highlighted.

2.3.1 Credits split rule

The higher is the total number N of credits of a course, the more essential are credits split rules for obtaining an applicable solution for that course classes. Without a split rule, credits can be spread in any way along the week, which implies in a significant higher number of possible combinations whenever N is big. Consequently the solution space increases, but more than that, this expansion may include solutions that are actually not interesting. Suppose the course "Math–3" has 7 credits to be spread along the week. It may not be interesting that these credits are split into 1 and 6 credits along two days, for example, or that all 7 credits are left in one single day.

The absence of credits split rules can therefore reduce solution quality, since it inserts a fake symmetry between solutions, and even hamper convergence while searching for the best solution, since it enlarges solution space.

2.3.2 Availability times

Probably the most important data quality issue is related to entities' availability times. As informed in 2.2.2, every professor, classroom and course has its own timetable availability. Also students have their possible assignment times constrained, as previously explained at "Student schedule".

Professors availabilities are usually the most hard and unstable information to be obtained, which is quite comprehensible. Many professors teach in more than one school or have extra activities, which implies in constrained and *floating* teaching availabilities. On the other hand, accuracy of these data is extremely important, since professors satisfaction is essential for automated timetables to be deployed. Inaccuracy may lead to professors assigned to classes at actually unavailable times or to the use of virtual teaching resources.

Most classrooms have a stable full-time availability, but some exceptions are common. Due to schools' physical space limitation, sometimes physical education classes take place at rented sport courts, which have therefore constrained availabilities. The opposite is also possible, i.e., that a set of classrooms of the institution is being used by external events on specific time slots.

Constraining time slots to which students are assigned is also essential. In schools the most common is that each student has classes restricted to a session of the day (Morning or Afternoon). Still, usually the total weekly workload increases for students of advanced grades, which can make it necessary to extend classes beyond their main session. The restriction in students availability must then be considered and is modeled through the concepts of shift and calender timetable, as previously explained. Mistakes in input data when associating a student demand to a shift and a calendar can lead to students assigned to classes at time slots that are out of their actual expected timetables.

Finally, because resources, courses and students can all have their availability limitations, it is clearly necessary that an appropriate intersection must exist between them so that demands are satisfied. For instance, inaccurate data could lead to a scenario where there are demands for the course "Biology-2" in the Afternoon, but professors capable of teaching this course are only available in the Morning. Such inconsistency would cause either an assignment with a virtual professor or non-satisfied demands.

2.3.3 Assignments between courses and rooms

As mentioned in 2.2.1, every course has a list of classrooms where it can be hold and an empty list is interpreted as there being no restrictions of rooms to where it can be assigned. Registering wrong assignments between courses and classrooms could lead to non applicable solutions. For example, one could incautiously register a computer training course as possible to be assigned to a biology laboratory instead to a computer lab, which would make no sense at all.

Gathering such data may not be a simple task though, specially when the institution has a large physical space. This is a particular harder issue for the university environment, where students are constantly moving between classrooms, courses are spread between several distinct departments and there is a high number of rooms. Fortunately, for schools this task is much less complicated and is usually done without great efforts.

3
Mathematical Formulation

The current method for solving the problem is based on integer linear programming (ILP) and the mathematical model used is described in this chapter.

Solving methods can be classified as exact or non-exact methods. Exact methods try to find proven optimal solutions, but might demand unrealistic amounts of processing time whether problem size increases. Non-exact methods, basically heuristics, are generally alternatives for searching for good enough solutions in a reasonable amount of time. Development of heuristics was a consequence of the lack for many years of efficient software tools for solution of integer programming models. They have been shown to be effective in practice and have thus received much attention from researches, as mentioned by (Santos et al., 2012). Since the early 90's though, advances in technology have allowed mathematical programming tools development, contributing then towards the usage of exact methods.

Section 3.1 introduces theory and main concepts of integer linear programming, which are fundamental in understanding the developed solving method. Further and more complete explanations about integer programming can be found at (Wolsey, 1998) and (Papadimitriou and Steiglitz, 1998).

Section 3.2 introduces the mathematical model developed for the problem.

3.1 Integer programming

A mathematical optimization problem consists of a set of inequations which together define feasible solutions for a problem and an objective function that guides the search for the best solution.

When both objective function and constraints are linear, then it is called *Linear Programming (LP).*

$$\text{Maximize } c \cdot x \tag{3.1a}$$

$$A \cdot x \leq b \tag{3.1b}$$

$$x \in \mathbb{R}^n_+ \tag{3.1c}$$

where A is a matrix such that $A \in \mathbb{R}^{m \times n}$, c is a row vector such that $c \in \mathbb{R}^n$, b is a column vector such that $b \in \mathbb{R}^m$, and x is a column vector of decision variables.

Variations of a linear program are obtained when one constraints variables' domain. A *Mixed Integer Program (MIP)* is a mathematical optimization in which some but not all variables are restricted to be integers.

$$\begin{array}{lccl} \text{Maximize} & c \cdot x & + & h \cdot y \\ & A \cdot x & + & G \cdot y \leq b \\ & x \in \mathbb{R}^n_+ & , & y \in \mathbb{Z}^p_+ \end{array}$$

where G is a matrix such that $G \in \mathbb{R}^{m \times p}$, h is a row vector such that $h \in \mathbb{R}^p$, and y is a column vector of integer decision variables.

When all decision variables must be integers, the program is called a *Pure Integer Linear Program (IP)*:

$$\text{Maximize } c \cdot x \tag{3.2a}$$

$$A \cdot x \leq b \tag{3.2b}$$

$$x \in \mathbb{Z}^n_+ \tag{3.2c}$$

When all decision variables are restricted to $0, 1$ values, then we have a *Binary Integer Program (BIP).*

Very efficient methods for solving linear programs are known, with the Simplex Method being the most famous one. Because integer programs look pretty much like linear programs, it is not a surprise that linear programming theory and practice is essential in understanding and solving integer programs. Solving an integer program is far more challenging than solving its linear version though.

Since the integer set $\mathbb{Z}$ is a countable set, it is clear that every integer program has a countable number of solutions and that every bounded integer program has a countable and finite set of solutions. Thus, theoretically, these problems can be solved by enumeration. In practice, to be feasible, such approach depends on the size of the possible solutions set, and that is where difficulty stems from.

To illustrate the situation, consider a general assignment problem, where n people are available to carry out n jobs and each person must be assigned to carry out exactly one job. There is an one-to-one correspondence between assignments and permutations of $1, \ldots, n$. Thus there are $n!$ solutions to compare. At table 3.1 one can have a glimpse of what happens in this case when n grows.

Table 3.1: Combinatorial explosion of possible solutions for factorial function problems

n	$n!$
10	$3.6 \cdot 10^{6}$
100	$9.33 \cdot 10^{157}$
1000	$4.02 \cdot 10^{2567}$

The obvious conclusion is that using complete enumeration for solving combinatorial problems, also known as “brute-force search” or “exhaustive search”, is feasible only for small values of n.

3.1.1 Alternative formulations

For every integer subset $X \subset \mathbb{Z}^n$ there is an infinite number of linear programs such that their feasible integer solution space is equal to X. Figure 3.1 illustrates a subset $X \subset \mathbb{Z}^2$ of integer points and 3 different possible ways (P_1, P_2 and P_3) to linearly constrain the subset so that no integer point of X is out of the bounded area and no integer point out of X is inside the bounded area.

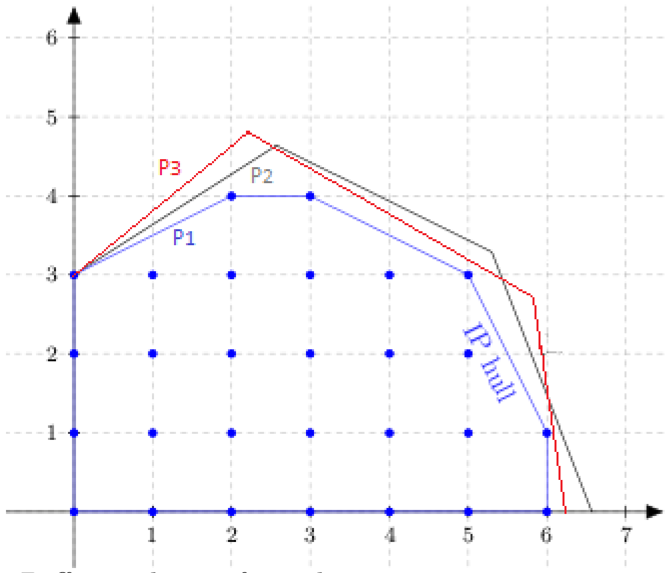

Figure 3.1: Different linear formulations

Following are some useful definitions for the subject.

Definition A subset of $\mathbb{R}^n$ described by a finite set of linear constraints $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a *polyhedron*.

Definition A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a *formulation* for a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ if and only if $X = P \cup (\mathbb{Z}^n \times \mathbb{R}^p)$.

Although all three formulations in figure 3.1 are valid for definition of subset X, the formulation P_1 is ideal, because solving a linear program over P_1 leads to an optimal solution lying at an extreme point. Since each extreme point of P_1 is integer, it follows that the integer program is solved.

Generally speaking, given a set $X \subseteq \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is a better formulation than P_2 if $P_1 \subset P_2$. Formulation P_1 is said to be *tighter* than P_2.

3.1.2 Optimality and relaxation

Regardless of the method used in the search for an optimal solution x^* of an integer program, some stopping criteria is needed in the algorithm, which is equivalent to prove that a given point x^* is optimal. A pretty intuitive way is to obtain lower and upper bounds for the objective function value of the optimal solution. Hence, given an IP

$$z = max\{c(x) : x \in X \subseteq \mathbb{Z}^n\},$$

any algorithm will construct a decreasing sequence

$$\overline{z_1} \geq \overline{z_2} \geq \ldots \overline{z_s} \geq z$$

of upper bounds, and an increasing sequence

$$\underline{z_1} \leq \underline{z_2} \leq \underline{z_t} \leq \ldots z$$

of lower bounds, and stop when $z_s - z_t$ is small enough.

Considering a maximization problem, any feasible solution provides a lower bound, also known as primal bounds. Finding upper bounds, known as dual bounds, is a different challenge. The main approach is by “relaxation”, the idea being to replace a difficult max (min) IP by a simpler optimization problem whose optimal value is at least as large (small) as z.

Definition A problem (RP) $z^R = max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ is a relaxation of (IP) $z = max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ if:

1. $X \subseteq T$, and

2. $f(x) \geq c(x)$ for all $x \in X$

Proposition If RP is a relaxation of IP, then $z^R \geq z$.

Linear programming relaxation

One way of relaxing an integer program is to enlarge the set of feasible solutions so that one optimizes over a larger set. For example, in figure 3.1 all the three formulations P_1, P_2 and P_3 are relaxations of the original problem if we ignore the integrality constraints.

Definition For the integer program $max\{c(x) : x \in P \cup \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$, the *linear programming relaxation* is the linear program $z^{LP} = max\{cx : x \in P\}$.

However, the more the feasible solution space is enlarged, the more different the relaxed problem becomes from the original problem, which can lead to more distinct optimal solutions values for both problems. When using linear relaxations for obtaining dual bounds, this is very relevant. The more relaxed in the objective function direction is the formulation, the more distant from the actual optimal solution value will probably be the upper bound obtained, as illustrated in figure 3.2. In general, the tighter is the relaxation, the better.

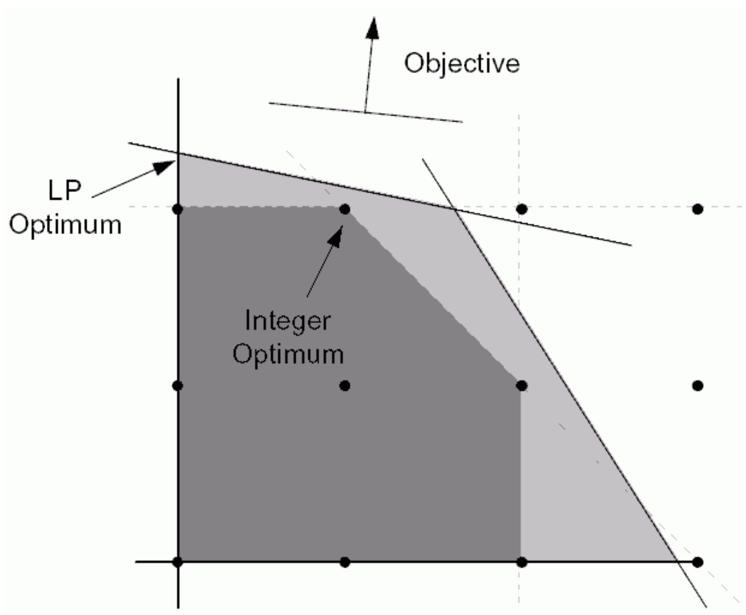

Figure 3.2: Linear relaxation in the direction of objective function

Notice, though, that for a relaxation of a IP to provide an upper bound, it must be solved to optimality.

Duality

Another way of finding dual bounds is through duality properties. Consider the previously introduced linear program given by (3.1), called *primal* problem, the following problem is referred as its *dual* problem.

$$\text{Minimize } u \cdot b \tag{3.3a}$$

$$u \cdot A \geq c \tag{3.3b}$$

$$u \in \mathbb{R}^m_+ \tag{3.3c}$$

Every linear program has an associated dual problem. The value of *any* dual feasible solution provides an upper bound on the primal objective function value, which makes it an advantage compared to the linear relaxation. A linear programming relaxation immediately leads to a dual problem, which can then be used to obtain dual bounds.

3.1.3 Integrality gap

Linear programming relaxation is thus a standard technique for designing approximation algorithms for optimization problems. Since such relaxation removes variables' integrality constraints, the optimal value for the relaxed maximization problem is necessarily greater than or equal to the optimal value for the original problem. The integrality gap is the maximum ratio between the solution quality of the integer program and of its relaxation. It essentially represents the inherent limits of a particular linear relaxation in approximating an integer program. Thus, the more the integral solution is improved, the smaller becomes the integrality gap. The tighter is the formulation, the smaller is the minimal gap.

3.1.4 Branch and Bound

Branch and bound is an implicit enumeration method for feasible solutions of combinatorial problems. Complete enumeration is inefficient and not applicable for real optimization problems, but using some knowledge on partition strategies of the problem it is possible to make intelligent decisions on whether it is necessary to explore a set of new potential solutions or about which set of potential solutions is more promising.

The "branch" word refers to the partitioning process of the problem and the "bound" refers to the decision of not exploring some set of possible new solutions (branch), avoiding exhaustive search. The bounding decisions use information of dual bounds to construct evidences of optimality and thus to eliminate branches that have proved to be unsuccessful.

Basically, for an integer linear program, the problem is decomposed and organized as a tree where each node is a subset of the solution space and a branch on that node splits the subset into mutually exclusive subsubsets. Each resulting subsubset in the partition is represented by a child of the original node. An algorithm based on linear programming is then used for calculating a dual bound on the cost of any solution in a given subset.

For further information about the branch and bound process and its strategies, see (Wolsey, 1998) and (Papadimitriou and Steiglitz, 1998).

3.1.5 MIP Solver

As previously mentioned, for decades limitation of computational resources was an obstacle to the development of efficient mathematical programming solvers. Since the early 90's though, advances in technology and algorithm design have allowed significant improvements in the area.

For mathematical programming there are some well-known solvers available. The most popular commercial software for optimization of (mixed integer) linear programming are IBM ILOG CPLEX Optimization Studio ((ILOG)), also simply known as Cplex, and Gurobi Optimizer ((Gurobi)). Both are accessible from several programming languages.

This work was developed using *Gurobi Optimizer version 6.0* for solving the mathematical models.

3.2 IP - Assignment Formulation for the Timetabling Problem

The general process of formulating an integer program can be organized in three steps:

1. Define what appear to be the necessary variables.
2. Use these variables to define a set of constraints so that the feasible points correspond to feasible solutions to the problem.
3. Use these variables to define the objective function.

Usually the need of auxiliary variables arises while formulating constraints, which gives an iterative feature to the process until a consistent model is reached.

Next subsections introduce the notation used in the mathematical model, which includes data sets, general data and model variables, and the formulation, which includes a multi-objective function and all constraints.

3.2.1 Notation

Sets

The basic sets used to describe the problem are:

- A - Set of all students. Elements of A are called a.
- D - Set of all courses. Elements of D are called d.
- I - Set of all course sections. Elements of I are called i.
- H - Set of all time slots. Elements of H are called h.
- Dt - Set of times of a day. Possible times are initial or ending moments of time slots. Elements of Dt are generally called dt, or dti for initial moments and dtf for ending moments.
- F - Set of all sessions of a day. Elements of F are called f.
- T - Set of all weekdays. Elements of T are called t.
- U - Set of all blocks. Elements of U are called u.
- S - Set of all classrooms. Elements of S are called s.
- P - Set of all professors. Elements of P are called p.

From the previous basic sets we define the following derived sets:

- $A_d \subseteq A$ - Set of students that required course $d \in D$.
- $Dm \subseteq D$ - Set of courses for which lessons are shareable between different classes, that is, lessons can be *merged*.
- $D_a \subseteq D$ - Set of courses required by student $a \in A$.
- $H_f \subseteq H$ - Set of time slots of session of day $f \in F$.
- $I_d \subseteq I$ - Set of sections of a course $d \in D$.
- $S_u \subseteq S$ - Set of classrooms of block $u \in U$.
- $P_{prior} \subseteq P$ - Set of all professors with priority level $prior$.

Model Data

Next, some data used in the model formulation are defined.

- Cap_s - Capacity of classroom s.
- $nCreds_d$ - Total of credits of course d.
- $duration_h$ - Total of minutes of time slot h.
- $duration_{hi,hf}$ - Total of minutes from the beginning of time slot hi to the end of time slot hf.
- M - called "big M", it is a "large enough value" that depends on the constraint it is applied to.
- $N_{d,k,t}$ - number of credits of the course d to be taught at day t if the credits split rule k is chosen. The value k is an integer identifier for the rule, ranging from 1 to K, where K is the total number of credits split rules for the course d.
- $NCH_{d,hi,hf}$ - number of credits for course d from time slot hi to time slot hf.
- $MinSize$ - minimum number of students required for offering a class. Default value is 1.
- $MaxSize_d$ - maximum number of students allowed in a section of course d.
- $delta_{t,f}$ - maximum of idle time allowed at phase f of day t for a professor. Used only for gap-constraints, usually is the average sum of intervals at phase f of day t of calenders.
- pv - Unique virtual professor, to be used whenever the set of real professors is not enough for demand satisfaction.
- $MaxNrCredsDay_{p,t}$ - maximum number of credits available at day t of professor p.
- $Time_{u1,u2}$ - minimum number of minutes necessary for moving between blocks $u1 \in U$ and $u2 \in U$. We consider as a long displacement a movement from $u1$ to $u2$ that takes more than 50 minutes.
- $MaxGapBetweenSessions$ - maximum idle interval between 2 consecutive day sessions that a professor is assigned to. It is used to avoid situations like a professor being assigned to a class early in the morning and then another just late in the afternoon, with a big idle inbetween interval. A reasonable value is around 170 minutes.

Variables

All variables used in the model are described as follows. The necessity of some of them depends on considering or not some constraints, in the sense that the more restrictions and requirements exist, the more auxiliary variables are needed.

- $v_{a,i,d,s,t,hi,hf}$ - binary variable, indicates if student a is assigned to a class of section i of course d, at classroom s, from time slot hi to time slot hf of day t.
- $x_{i,d,u,s,t,hi,hf}$ - binary variable, indicates if section i of course d is assigned to classroom s from time slot hi to time slot hf of day t.
- $z_{i,d}$ - binary variable, indicates if section i of course d is offered.
- $o_{i,d,s}$ - binary variable, indicates if section i of course d is offered at classroom s.
- $fd_{d,a}$ - binary slack variable, indicates if student a has his request for course d not satisfied.
- $m_{d,i,k}$ - binary variable, indicates if the credits split rule k was chosen for section i of course d.
- $s_{i,d,a}$ - binary variable, indicates if student a is assigned to section i of course d.
- $k_{p,i,d,u,t,h}$ - binary variable, indicates if professor p teaches to section i of course d at block u at day t and time slot h.
- $y_{p,i,d}$ - binary variable, indicates if professor p is assigned to section i of course d.
- $hip_{p,t,f}$ - integer variable, indicates the time in minutes of the first time slot assigned to professor p at session f of day t. For example, if the first class assigned to p on $t = Monday$ and $f = morning$ starts at 9:30 am, then $hip_{p,t,f} = 9 \cdot 60 + 30 = 570$.
- $hfp_{p,t,f}$ - integer variable, indicates the time in minutes of the end of the last time slot assigned to professor p at session f of day t.
- $fpgap_{p,t,f}$ - integer slack variable, indicates the gap in session f of day t of professor p's schedule.
- $begin_{a,t,h}$ - binary variable, indicates if first class of student a at day t begins at time h.
- $end_{a,t,h}$ - binary variable, indicates if last class of student a at day t ends at time h.

- $at_{a,t}$ - binary variable, indicates if student a has classes at day t.
- $uu_{p,t,u}$ - binary variable, indicates if professor p has classes at day t in block u.
- $displac_{p,t,u1,u2}$ - binary variable, indicates whether professor p has classes at day t in block $u1$ and in block $u2$. In practice, it identifies that at least one displacement is made between the both blocks in the day.
- $ptf_{p,t,f}$ - binary variable, indicates if professor p has classes at session f of day t.
- $pt_{p,t}$ - binary variable, indicates if professor p has classes at day t.

3.2.2 Formulation

Objective function

The following general objective function includes all possible minimization goals, with their subjective weights to be decided according to the scenario or the user preferences. Further alternatives of how to handle disparate goals are discussed in Section 4.2.

$$\begin{aligned}
\text{MIN} \quad & \lambda \cdot \sum_{a \in A} \sum_{d \in D_a} \cdot fd_{d,a} \\
& + \theta \cdot \sum_{i \in I_d} \sum_{d \in Dm} z_{i,d} \\
& + \epsilon \cdot \sum_{p \in P} \sum_{t \in T} \sum_{f \in F} fpgap_{p,t,f} \\
& + \sigma \cdot \sum_{p \in P} \sum_{t \in T} \sum_{u1 \in U} \sum_{u2 \in U} Time_{u1,u2} \cdot displac_{p,t,u1,u2} \\
& + \gamma \cdot \sum_{i \in I_d} \sum_{d \in D} y_{pv,i,d} \\
& + \alpha \cdot \sum_{p \in P} \sum_{t \in T} pt_{p,t} \\
& + \alpha \cdot \sum_{p \in P} \sum_{t \in T} \sum_{f \in F} ptf_{p,t,f}
\end{aligned}$$

Assigns 'v' to 'x' and ensures classroom capacity

$$\sum_{a \in A} v_{a,i,d,s,t,hi,hf} \leq Cap_s \cdot x_{i,d,s,t,hi,hf} \tag{3.4}$$

$$\forall d \in D_a \quad \forall i \in I_d \quad \forall s \in S \quad \forall t \in T \quad \forall hi \in H \quad \forall hf \in H$$

Assigns the classroom of a section to variable 'o'

$$M \cdot o_{i,d,s} \geq \sum_{t \in T} \sum_{hi \in H} \sum_{hf \in H} x_{i,d,s,t,hi,hf} \tag{3.5}$$
$$\forall d \in D \quad \forall i \in I_d \quad \forall u \in U \quad \forall s \in S_u$$

Ensures single classroom per course section

$$\sum_{u \in U} \sum_{s \in S_u} o_{i,d,s} = z_{i,d} \qquad \forall d \in D \quad \forall i \in I_d \tag{3.6}$$

No overlapping in classroom's timetable

$$\sum_{i \in I_d} \sum_{d \in D} \sum_{hi \in H_d} \sum_{\substack{hf \in H_d \\ hi \leq hf}} x_{i,d,s,t,hi,hf} \leq 1 \tag{3.7}$$
$$\forall u \in U \quad \forall s \in S_u \quad \forall t \in T \quad \forall h \in H \quad (hi, hf) \text{ overlaps } h$$

Tries to satisfy each student requirement

$$\sum_{i \in I_d} s_{i,d,a} + fd_{d,a} = 1 \qquad \forall a \in A \quad \forall d \in D_a \tag{3.8}$$

Ensures a minimum number of students in each course section

$$\sum_{a \in A_d} s_{i,d,a} \geq MinSize \cdot z_{i,d,cp} \qquad \forall d \in D \quad \forall i \in I_d \tag{3.9}$$

Ensures a maximum number of students in each course section

$$\sum_{a \in A_d} s_{i,d,a} \leq MaxSize_d \cdot z_{i,d,cp} \qquad \forall d \in D \quad \forall i \in I_d \tag{3.10}$$

Assigns student's course section to lessons and ensures course section's total number of credits

$$nCreds_d \cdot s_{i,d,a} = \sum_{s \in S} \sum_{t \in T} \sum_{hi \in H} \sum_{hf \in H} NHC_{d,hi,hf} \cdot v_{a,i,d,s,t,hi,hf} \qquad \forall a \in A \quad \forall i \in I_d \quad \forall d \in D_a \tag{3.11}$$

Links variables x and variable z and ensures course section's total number of credits

$$\sum_{s \in S} \sum_{t \in T} \sum_{hi \in H} \sum_{hf \in H} NHC_{d,hi,hf} \cdot x_{i,d,s,t,hi,hf} = nCreds_d \cdot z_{i,d} \qquad \forall i \in I_d \quad \forall d \in D \tag{3.12}$$

Avoids classes overlapping at student timetable

$$\sum_{u \in U} \sum_{s \in S_u} \sum_{i \in I_d} \sum_{d \in D_a} \sum_{hi \in H_d} \sum_{\substack{hf \in H_d \\ hi \leq hf \\ (hi,hf) \text{ overlaps } h}} v_{a,i,d,s,hi,hf,t} \leq 1 \qquad \forall a \in A \quad \forall t \in T \quad \forall h \in H_d \tag{3.13}$$

Single lesson for each course section per day

$$\sum_{u \in U} \sum_{s \in S_u} \sum_{hi \in H_d} \sum_{\substack{hf \in H_d \\ hi \leq hf}} x_{i,d,s,t,hi,hf} \leq 1 \qquad \forall d \in D \quad \forall i \in I_d \quad \forall t \in T \tag{3.14}$$

Credits split rules

Credits split rule for each course section

$$\sum_{u \in U} \sum_{s \in S_u} \sum_{hi \in H_d} \sum_{\substack{hf \in H_d \\ hi \leq hf}} NCH_{d,hi,hf} \cdot x_{i,d,u,s,t,hi,hf} = \sum_{k \in K_d} N_{d,k,t} \cdot m_{d,i,k} \qquad \forall d \in D \quad \forall i \in I_d \quad \forall t \in T \tag{3.15}$$

Single credits split rule for each course section

$$\sum_{k \in K_d} m_{d,i,k} = z_{i,d} \qquad \forall d \in D \quad \forall i \in I_d \tag{3.16}$$

Student's gaps prohibition

In the following, auxiliary variables necessary for preventing gaps in student's timetable are consistently set.

Sets if student a has classes at day t (variable $at_{a,t}$)

$$M \cdot at_{a,t} \geq \sum_{i \in I_d} \sum_{d \in D_a} \sum_{u \in U} \sum_{s \in S_u} \sum_{hi \in H_d} \sum_{\substack{hf \in H_d \\ hi \leq hf}} v_{a,i,d,u,s,t,hi,hf} \qquad \forall a \in A \quad \forall t \in T \tag{3.17}$$

Sets starting time of classes for student a at day t (variable $begin_{a,t,h}$)

$$\begin{aligned} begin_{a,t,h} = & \sum_{i \in I_d} \sum_{d \in D_a} \sum_{u \in U} \sum_{s \in S_u} \sum_{hf \in H_d} v_{a,i,d,u,s,t,h,hf} \\ & - \sum_{h' \leq h} begin_{a,t,h'} + \sum_{h' \leq h} end_{a,t,h'} \\ & \forall a \in A \quad \forall t \in T \quad \forall h \in H \end{aligned} \tag{3.18}$$

Sets ending time of classes for student a at day t (variable $end_{a,t,h}$)

$$\begin{aligned} end_{a,t,h} = & \sum_{i \in I_d} \sum_{d \in D_a} \sum_{u \in U} \sum_{s \in S_u} \sum_{hi \in H_d} v_{a,i,d,u,s,t,hi,h} \\ & - \sum_{h' \geq h} end_{a,t,h'} + \sum_{h' \geq h} begin_{a,t,h'} \\ & \forall a \in A \quad \forall t \in T \quad \forall h \in H \end{aligned} \tag{3.19}$$

Uniqueness of student's last class of the day

$$\sum_{h \in H} end_{a,t,h} = at_{a,t} \qquad \forall a \in A \quad \forall t \in T \tag{3.20}$$

Uniqueness of student's first class of the day

$$\sum_{h \in H} begin_{a,t,h} = at_{a,t} \qquad \forall a \in A \quad \forall t \in T \tag{3.21}$$

Next, gaps in student timetable are prevented. Unlike professors similar restriction, students cannot have idle times, and therefore these are hard constraints and there are no slack variables.

Prohibits gap in student timetable

$$\sum_{i \in I_d} \sum_{d \in D_a} \sum_{s \in S} \sum_{hf \in H} v_{a,i,d,s,t,hi_+,hf} = \tag{3.22}$$
$$\sum_{i \in I_d} \sum_{d \in D_a} \sum_{s \in S} \sum_{hf \in H} v_{a,i,d,s,t,hi_-,hf} - end_{a,t,hi_-} + begin_{a,t,hi_+}$$
$$\forall a \in A \quad \forall t \in T \quad \forall hi_+ \in H \quad hi_- \in H \text{ s.t. } hi_- + 1 = hi_+$$

Professors assignment

The following constraints, responsible for assigning professors to lessons, involve both real professors and the virtual teaching resource pv.

No overlapping in professor's timetable

$$\sum_{\substack{dti,dtf \in Dt \\ dt \in [dti,dtf)}} \sum_{i \in I_d} \sum_{d \in D} \sum_{u \in U} k_{p,i,d,u,t,dti,dtf} \leq 1 \tag{3.23}$$
$$\forall p \in P \cup \{pv\} \quad \forall t \in T \quad \forall dt \in Dt$$

Assigns professor to lessons (sets variable k)

$$\sum_{\substack{hi,hf \in H \\ dti \in [hi,hf)}} \sum_{s \in S_u} x_{i,d,s,t,hi,hf} \leq k_{p,i,d,u,t,dti} + (1 - y_{p,i,d}) \tag{3.24}$$
$$\forall i \in I \quad \forall d \in D \quad \forall u \in U \quad \forall p \in P \cup \{pv\} \quad \forall t \in T \quad \forall dti \in Dt$$

$$\sum_{\substack{hi,hf \in H \\ dti \in [hi,hf)}} \sum_{s \in S} x_{i,d,s,t,hi,hf} \leq \sum_{u \in U} \sum_{p \in P \cup \{pv\}} k_{p,i,d,u,t,dti} \tag{3.25}$$
$$\forall i \in I \quad \forall d \in D \quad \forall t \in T \quad \forall dti \in Dt$$

Assigns professor to course section (sets variable y)

$$\sum_{u \in U} \sum_{t \in T} \sum_{h \in H} k_{p,i,d,u,t,h} = nCreds_d \cdot y_{p,i,d} \qquad (3.26)$$
$$\forall i \in I \quad \forall d \in D \quad \forall p \in P \cup \{pv\}$$

Assigns a single professor to each course section

$$\sum_{p \in P \cup \{pv\}} y_{p,i,d} = z_{i,d} \qquad \forall i \in I \quad \forall d \in D \qquad (3.27)$$

Professor's displacement

In this subsection quality constraints related to displacement of real professors along the week and in a day are considered.

Ensures enough time for displacement between blocks in the same day

$$\sum_{i \in I_d} \sum_{d \in D} k_{p,i,d,u1,t,h1} + \sum_{i \in I_d} \sum_{d \in D} k_{p,i,d,u2,t,h2} \leq 1 \qquad (3.28)$$
$$\forall p \in P \quad \forall t \in T \quad \forall u1 \in U \quad \forall u2 \in U \quad \forall h1 \in H \quad \forall h2 \in H$$

s.t. time between ending of h1 and beginning of h2 is less than displacement time between u1 and u2.

Ensures the maximum of 1 displacement for each professor in the same day

$$\sum_{i \in I_d} \sum_{d \in D} k_{p,i,d,u1,t,h1} + \sum_{i \in I_d} \sum_{d \in D} k_{p,i,d,u2,t,h2} + \sum_{i \in I_d} \sum_{d \in D} k_{p,i,d,u3,t,h3} \leq 2 \qquad (3.29)$$
$$\forall p \in P \quad \forall t \in T \quad \forall u2 \in U \quad \forall h2 \in H$$
$$\text{s.t. } u1 \neq u2 \qquad u3 \neq u2 \qquad h1 < h2 \qquad h3 > h2$$

Identifies if the professor teaches in a block at the day and ensures the maximum daily number of credits for the professor

$$\sum_{i \in I_d} \sum_{d \in D} \sum_{h \in H} k_{p,i,d,u,t,h} \leq MaxNrCredsDay_{p,t} \cdot uu_{p,t,u} \qquad (3.30)$$
$$\forall p \in P \quad \forall t \in T \quad \forall u \in U$$

Constraints the maximum number of blocks assigned to the professor at the day

$$\sum_{u \in U} uu_{p,t,u} \leq 2 \qquad \forall p \in P \quad \forall t \in T \tag{3.31}$$

Identifies 2 distinct blocks being assigned to the same day of the professor

$$uu_{p,t,u1} + uu_{p,t,u2} \leq 1 + displac_{p,t,u1,u2} \tag{3.32}$$
$$\forall p \in P \quad \forall t \in T \quad \forall u1 \in U \quad \forall u2 \in U \quad \text{s.t. } u1 \neq u2$$

Limits the maximum of 2 distant blocks assigned to the same day of a professor

$$\sum_{u1 \in U} \sum_{\substack{u2 \in U \\ Time_{u1,u2} \geq 50}} displac_{p,t,u1,u2} \leq 1 \qquad \forall p \in P \quad \forall t \in T \tag{3.33}$$

Limits the maximum of 2 days of the week that a professor can have a big displacement

$$\sum_{t \in T} \sum_{u1 \in U} \sum_{\substack{u2 \in U \\ Time_{u1,u2} \geq 50}} displac_{p,t,u1,u2} \leq 2 \qquad \forall p \in P \tag{3.34}$$

Professor's gaps avoidance

Next, variables $hip_{p,t,f}$ and $hfp_{p,t,f}$ are set. We draw attention to the fact of these variables being strictly set, i.e., less and greater inequality constraints are used per variable, resulting in an implicit equality constraint. Consequently, they do not depend on an objective function, which is important when it has conflicting goals. The only exception is when the professor is not assigned at the day — in this case $hip_{p,t,f}$ assumes the value of the latest time slot of the day session and $hfp_{p,t,f}$ assumes analogously the value of the earliest time slot of the day session, as one can infer from the constraints below.

Sets variable $hip_{p,t,f}$

$$hip_{p,t,f} \geq m(dt) \cdot (1 - \sum_{k \in K_{dti<dt}} k_{p,t,dti}) \tag{3.35}$$
$$\forall p \in P \quad \forall t \in T \quad \forall f \in F \quad \forall dt \in Dt_f$$

$$hip_{p,t,f} \leq m(dt) + M \cdot (1 - \sum_{k} k_{p,t,dti}) \tag{3.36}$$
$$\forall p \in P \quad \forall t \in T \quad \forall f \in F \quad \forall dti \in Dt_f$$

Sets variable $hfp_{p,t,f}$

$$hfp_{p,t,f} \geq \sum_{k} m(dt) \cdot k_{p,t,dtf} \tag{3.37}$$
$$\forall p \in P \quad \forall t \in T \quad \forall f \in F \quad \forall dtf \in Dt_f$$

$$hfp_{p,t,f} \leq m(dt) + M \cdot (\sum_{k \in K_{dtf > dt}} k_{p,t,dtf}) \tag{3.38}$$
$$\forall p \in P \quad \forall t \in T \quad \forall f \in F \quad \forall dt \in Dt_f$$

Next, gaps in professor timetable are controlled. The usage of the slack variable $fpgap_{p,t,f}$ indicates the amount of idle time in session f of day t for professor p. The reduction of professors idle time is achieved through minimization of these slack variables in objective function.

Prohibits gap in each session of day in professor timetable

$$\sum_{k \in K_{h \in H_f}} duration_h \cdot k_{p,t,h} + delta_{f,t} + fpgap_{p,t,f} \geq hfp_{p,t,f} - hip_{p,t,f} \tag{3.39}$$
$$\forall p \in P \quad \forall t \in T \quad \forall f \in F$$

Professor's timetable compactness

Besides the constraints introduced above, specific for avoiding gaps in each session of a professor's day, further quality constraints related to compactness of real professors' timetable along the week and in a day are considered.

Sets if a session of day is assigned for the professor

$$M \cdot ptf_{p,t,f} \geq \sum_{d \in D} \sum_{i \in I_d} \sum_{u \in U} \sum_{h \in H_f} k_{p,i,d,u,t,h} \tag{3.40}$$
$$\forall p \in P \quad \forall T \in T \quad \forall f \in F$$

Sets the number of days of the week assigned to each professor

$$M \cdot pt_{p,t} \geq \sum_{f \in F} ptf_{p,t,f} \qquad \forall p \in P \quad \forall t \in T \tag{3.41}$$

Prohibits big gaps between assigned sessions of the day

$$\begin{aligned} hip_{p,t,f} - hfp_{p,t,f-1} \leq MaxGapBetweenSessions \\ +M \cdot (2 - ptf_{p,t,f} - ptf_{p,t,f-1}) \\ \forall p \in P \quad \forall t \in T \quad \forall f \in F \end{aligned} \tag{3.42}$$

4
Solving Strategies

This chapter describes and compares different strategies and approaches used and tested during solver development.

4.1
Phases of problem solving

In section 3.2 the complete linear program used as basic foundation for solving the problem is introduced. From practice, though, arose the need of split the problem into some phases, so that better solutions could be found.

The first partition is related to virtual teaching resources. Virtual professors can be interpreted as the necessity of hiring professors, and should then be used only when there is no real professor available. It makes sense to consider virtual resources apart from the initial problem, that is, first to solve a model with only real resources and then solving a second model that allows virtual resources only for those demands not satisfied at first stage. The only obstacle to such partition is whether gaps in student's timetable are directly prohibited by the model (see constraints at section 3.2.2). In this scenario, by partitioning the problem, any case of non-satisfied demand with a real professor will generate an empty time slot at an extremity of a day, but never between allocated classes. The problem is that this could easily prevent satisfaction of other demands when professor availability is restricted, which is very common.

As a simple example, suppose that due to a lack of teaching resource a demand of some class for Mathematics lessons will inevitably be non-satisfied with a real professor. The History professor, Norah Jones, can only teach at 08:00 on Monday morning. If lessons have one-hour duration, and if due to professors limited availability no other course can be assigned at 9:00 am on Monday, but only later, then either an one-hour gap will be generated from 9:00 to 10:00 am and later filled in by a Math lesson with a virtual resource, or Prof. Jones will be prevented for teaching the History lesson to the class, so that a gap is not created. In case of students' gaps constraints are being considered, necessarily only the second option would last, which is actually not interesting, since a History demand would be not satisfied due to inconsistency of input data for Mathematics teaching resource. This small example gives us a glimpse of how important is input data consistency and how aspects not obviously related at first sight are actually connected.

Thus, splitting the problem into real and virtual resources phases is advised only when students' gaps constraints have not been included in the linear program. Fortunately, if student's available timetable fits exactly his courses demands, there is no need of such constraints.

The second partition refers to professors priorities.

Next subsections detail the partition problem supposing that students' gaps constraints are dispensable.

4.1.1 Real teaching resource

As pointed out while introducing the concept of professor at section 2.2.1, professors might have different importance levels, which means that a professor can have his assignments prioritized over others. Such distinction emerged from practice, when analyzing some possible solutions to the timetabling problem of a Brazilian school. What happens is that, as discussed in section 1.5, actual timetabling for educational institutions is a political just as much as a scheduling problem. It might be acceptable if the timetable of a fresh professor of the school does not end up as compact as it would be desired, but it is unacceptable that timetable of an old professor of the school, considered as a loyal employee, does not respect a minimum and high quality level.

Real teaching resources should then receive different treatment depending on their priorities. The best approach depends on how much higher priority features can be sacrificed so that overall features are satisfied. Given that, we suggest 2 possible approaches:

Split If professors priorities are strictly important, one should avoid as much as possible that lower-priority professors requirements interfere with higher ones, even if it implies that the overall solution has lower quality *in some goal.*

1. **First Priority** We consider, as the first phase of the problem solving process, the solving of a MIP containing only real and first priority professors. This avoids that particularities and restrictions on lower-priority professors interfere with higher-priority professors assignments.

2. **Second Priority** In the second phase of the problem solving process we solve a MIP containing only real professors, with all priority levels, but ensuring timetable quality for first priority professors reached at solution of the first phase.

Merged If professors priorities are not strictly important, the best might be to consider all real professors in a single phase, but with goal weights that vary according to professor priority. This means, for example, it may be worthy not to satisfy a request for a higher-priority professor whether it implies in satisfying some request for 2 lower-priority professors. Such approach is simpler and generally faster, but depends on subjective weights choices and can result in solutions that are harder to analyze.

4.1.2 Virtual teaching resource

After solving the problem using all real resources available, if there is any non-satisfied demand, it is time to consider virtual teaching resources. The best solution found with real resources at previous phase is fixed and virtual professors are used only to satisfy the remaining demands, filling in empty time slots in students timetables.

Maximizing remaining demand satisfaction through the best use of virtual resources

Supposing that the best solution found until this moment has been fixed and thus prevented from being modified, then the last thing to do is to satisfy the maximum of remaining demands with virtual resources.

Since virtual teaching resources have neither quality nor overlapping constraints, they are very easy assignments and could even be done as a post-processing. Still, we chose to make use of mathematical programming to do it, for 2 reasons: first, the model formulation and implementation is just the same one we have been using, except that current real solution must be fixed and virtual resources allowed; second, we can take advantage of mathematical programming to easily remove some symmetry around virtual resources usage.

It is very common for professors to restrict their available teaching time. Suppose the Biology professor, Glen Hansard, is only available on Tuesdays and Fridays, and the Geography professor, Freddie Mercury, is only available on Wednesdays and Fridays. Besides, suppose that due to a lack of teaching resources, a virtual professor must be assigned to the Biology and Geography lessons of class $C1$. For purposes of simplification, consider that both Biology and Geography courses have 1 credit each. Class $C1$'s timetable must have at least 2 empty time slots then, otherwise its demand would be greater than it can support. Finally, suppose that assignments of other courses attended by $C1$ could be arranged so that we have the following possibilities:

– an empty time slot on Monday and on Wednesday, or

– an empty time slot on Tuesday and on Wednesday.

Now, the decision to be made is to which time slot assign the Biology and the Geography virtual lessons. Are they equally good? Possible solutions are:

1. assign Biology to Monday and Geography to Wednesday, or

2. assign Biology to Tuesday and Geography to Wednesday, or

3. assign Geography to Monday and Biology to Wednesday.

4. assign Geography to Tuesday and Biology to Wednesday, or

The difference is that, in case of using virtual teaching resource, assigning the course lesson to a time slot at which no capable professor has registered availability implies either hiring a new professor for teaching the course or negotiating with the current capable professors their availabilities. On the other hand, assigning the course lesson to a time slot at which a capable professor has registered availability necessarily implies hiring a new professor, since if a capable professor is already registered at that time slot and still the lesson is being taught by a virtual resource, it means either that the professor was assigned to another class at that moment or that he has been blocked by some quality constraint.

Bearing this in mind, best options for virtual assignment in the previous example are 3 and 4, because neither Prof. Hansard is available on Wednesday nor Prof. Mercury is available on Monday and Tuesday, the intermediate option is 1, and the worst option is 2, since days chosen for courses classes are exactly the days respectively already available for both professors.

Objective function for virtual resource phase To simplify notation, we define KV_d as being a set containing all variables k of virtual professor for course d. Such set can be split into two subsets according to professors availabilities, let us say $KVPA_d$ and its complement $\overline{KVPA}_d$, so that $KVPA_d \cup \overline{KVPA}_d = KV_d$, where $KVPA_d$ is the subset of all variables k of virtual professor for course d and for a time slot at which some real capable professor has registered availability.

Then, what we aim is to satisfy the remaining demand using virtual teaching resources, while giving preference to those assignments that reduce the certainty of having to hire new professors and guaranteeing the current solution. Since virtual resource phase is much simpler and we do not have conflicting goals, an unified objective function works just fine and is enough. This last goal is achieved by:

$$\text{g} = \text{MIN } (\sum_{a \in A} \sum_{d \in D} fd_{d,a} + 0.00001 \cdot \sum_{k \in KVPA_d} k_{pv,i,d,u,t,h})$$

subjected to assignments made in the best
solution found when considering real resources

4.2 Goal programming

Just as in any other kind of problem, the feasibility of a linear model for a timetabling problem depends utterly on input data set. In general, there is no guarantee that all demanded requirements can be satisfied with the specified available resources, even because usually there are several and conflicting objectives. Therefore, specially in a commercial solver, it is important to treat some idealistic hard constraints as highest priority soft constraints, otherwise the methodology could frequently generate an infeasible model.

In real life it is useless for an educational institution to have a solution that satisfies all demands of students for courses, but which does not respect some important quality requirements. Bearing this in mind, in addition to the feasibility issue, the model presented in this work, unlike most authors, does not consider demand satisfaction as a hard constraint. Essential quality requirements are modeled as hard constraints though, and demand satisfaction is treated as the highest priority objective, followed by some further goals.

A **multi-objective function** unifies disparate goals of the model in a single weighted sum of preferences. Such approach, although simpler, depends on a very subjective choice of weights and is not always consistently reliable, specially if involving trade-off, when objectives are mutually conflicting. It can be very difficult to quantify quality.

An alternative approach is using a priority line for the different goals. According to the institution preferences, one optimizes the problem in a sequence of steps, where each step is responsible for a goal, from the most to the least important. For each step a different and specific objective function is used and the feasible solution space is subject to features of the best solution found at the previous step. Such approach is known as **preemptive goal programming** and is following detailed.

4.2.1 General concept

A goal programming model seeks to simultaneously take into account several objectives or goals that are concern to a decision maker. While a linear programming model consists of constraints and a single objective function to be maximized or minimized, a goal programming model consists of constraints and a set of goals that are prioritized in some sense. In both linear and goal programming problems, if the constraints are inconsistent, there are no feasible solutions for the model. In goal programming, however, one can expect that although there is a set of feasible solutions satisfying the constraints, possibly none of them satisfies simultaneously all the conflicting goals of the organization. The objective of goal programming is to find a solution that satisfies the true constraints and comes closest to meeting the stated goals.

In **lexicographic** or **preemptive goal programming** the decision maker orders the unwanted deviations into a number of priority levels, with the minimization of a deviation in a higher priority level being infinitely more important than any deviations in lower priority levels. A lexicographic goal program can be solved as a series of linear programs and should be used when there is a clear priority ordering amongst the goals to be achieved. The idea behind the preemptive goal programming approach is that lower priority level goals should not be attained at the expense of higher priority goals — they are preempted.

If the decision maker is more interested in direct comparisons of the objectives then weighted or **nonpreemptive goal programming** should be used. In this case all the unwanted deviations are multiplied by weights, reflecting their relative importance, and added together as a single sum to form the achievement function, as done in the last section. This converts the goal programming model into a linear programming model. Güenalay and Sahin use goal programming at (Guenalay and Sahin, 2006) to satisfy instructors' preferences as much as possible. More information at (Unknown).

4.2.2 Applying Goal Programming

Next are presented the possible goal programming approaches for a real resource phase of the problem.

Nonpreemptive Goal Programming

Following there is the unified objective function considered at this work for the real resource phase. Note that there is no variable controlling virtual professor usage. Although here it is very easy to change priorities order and adding or removing goals, there is the necessity of choosing coefficients for variables at objective function.

$$\begin{aligned}\text{MIN}\quad & \lambda \cdot \sum_{a\in A}\sum_{d\in D} \cdot fd_{d,a} \\ & +\epsilon \cdot \sum_{p\in P}\sum_{t\in T}\sum_{f\in F} fpgap_{p,t,f} \\ & +\sigma \cdot \sum_{p\in P}\sum_{t\in T}\sum_{u1\in U}\sum_{u2\in U} Time_{u1,u2} \cdot displac_{p,t,u1,u2} \\ & +\theta \cdot \sum_{i\in I_d}\sum_{d\in Dm} z_{i,d} \\ & +\alpha \cdot \sum_{p\in P}\sum_{t\in T}(pt_{p,t} + \sum_{f\in F} ptf_{p,t,f})\end{aligned}$$

An unified objective function is more appropriate when goals have not a strict priority order, in the sense that their combinations can produce solutions that are Pareto efficient. So, the first question when choosing the approach is whether one wants to order or to weight goals.

Preemptive Goal Programming

Following are listed the goals, ordered by priority, considered in this work for the real resource phase when using preemptive goal programming. For each step a MIP is solved with a specific objective function and the respective feasible solution space is additionally constrained by the solution of the previous phase (figure 4.1 illustrates this encapsulation). We draw attention to the nonexistence of coefficients multiplying variables at each objective function and to the ease of changing priorities order and adding or removing goals.

1. Maximizing demand satisfied by real professors

$$\text{g1} = \text{MIN} \sum_{a\in A}\sum_{d\in D} fd_{d,a}$$

2. Minimizing the number of sections, in case of courses that can have classes merged

$$\text{g2} = \text{MIN} \sum_{i\in I_d}\sum_{d\in Dm} z_{i,d}$$

$$\sum_{a\in A}\sum_{d\in D} fd_{d,a} \leq g1$$

Where the additional constraint restricts the search-space to solutions where the satisfied demand is no less than the value $g1$ achieved at previous step.

3. Minimizing real professor displacement

$$\text{g3} = \text{MIN} \sum_{p \in P} \sum_{t \in T} \sum_{u1 \in U} \sum_{u2 \in U} Time_{u1,u2} \cdot displac_{p,t,u1,u2}$$
$$\sum_{a \in A} \sum_{d \in D} fd_{d,a} \leq g1$$
$$\sum_{i \in I_d} \sum_{d \in Dm} z_{i,d} \leq g2$$

Where the additional constraint restricts the number of sharable classes to be no more than the minimal value $g2$ achieved at previous step.

4. Minimizing number of days and sessions of day used in real professor's timetable

$$\text{g4} = \text{MIN} \sum_{p \in P} \sum_{t \in T} (pt_{p,t} + \sum_{f \in F} ptf_{p,t,f})$$
$$\sum_{a \in A} \sum_{d \in D} fd_{d,a} \leq g1$$
$$\sum_{i \in I_d} \sum_{d \in Dm} z_{i,d} \leq g2$$
$$\sum_{p \in P} \sum_{t \in T} \sum_{u1 \in U} \sum_{u2 \in U} Time_{u1,u2} \cdot displac_{p,t,u1,u2} \leq g3$$

Where the additional constraint restricts the solution space to the minimum real professor displacement achieved at previous step.

5. Minimizing gaps in real professor's timetable

$$\text{g5} = \text{MIN} \sum_{p \in P} \sum_{t \in T} \sum_{f \in F} fpgap_{p,t,f}$$
$$\sum_{a \in A} \sum_{d \in D} fd_{d,a} \leq g1$$
$$\sum_{i \in I_d} \sum_{d \in Dm} z_{i,d} \leq g2$$

$$\sum_{p\in P}\sum_{t\in T}\sum_{u1\in U}\sum_{u2\in U} Time_{u1,u2}\cdot displac_{p,t,u1,u2} \leq g3$$

$$\sum_{p\in P}\sum_{t\in T}(pt_{p,t}+\sum_{f\in F} ptf_{p,t,f}) \leq g4$$

Where the additional constraint restricts the solution space to the minimum number of days and days session used in real professors timetables achieved at previous step.

The final solution for the real resource phase is the one with cost equal to $g5$, achieved at last stage. If an optimal solution is achieved for all steps, then at the end of the last step we guarantee that the final solution found is the best one with respect to the order of priorities.

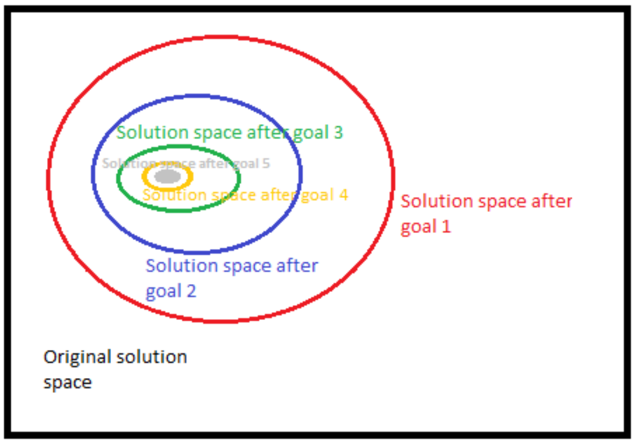
Solution space after goal 3
Solution space after goal 5
Solution space after goal 4
Solution space after goal 1
Solution space after goal 2
Original solution space

Figure 4.1: Solution spaces are limited and encapsulated after each goal optimization step.

Through preemptive goal programming it is easier to understand and evaluate the best solution found by the optimization process and the reasons of non-satisfaction of different goals, especially when the number of goals to be achieved through the objective function increases. On the other side, it implies more steps and consequently a possible increase of run time.

4.3 Polishing Method

For real-world school timetabling, where problem size is not small, solving the whole MIP introduced in Section 3.2 of Chapter 3 at once with a MIP solver like Gurobi or Cplex has been shown to be ineffective and unworkable. Tests made with schools with an ordinary size (maximal $\approx$ 30 classes, $\approx$ 700 demanded credits, $\approx$ 1000 teaching available time slots and $\approx$ 2 blocks) have shown that generic MIP solvers find it already difficult to converge to a good integer solution. For larger schools, both the linear relaxation and the branch and bound phase are *very* hard and have proven themselves impossible to be solved by current MIP solvers.

An auxiliary method was then developed for helping convergence while solving the MIP. It is described at next sections and referred here as a “polishing method”.

4.3.1 The general idea

Given an lp model and an initial feasible solution s, the idea of a polishing method is iteratively to solve models lp' similar but easier than the original model, such that any feasible solution for lp' is also a feasible solution for lp and the solution at the end of each iteration is always at least as good as the one at the beginning. In another words, it is generated a sequence of feasible solutions with monotonically (but not strictly) increasing quality.

Suppose that a solution $s \in \mathbb{Z}^n$ is feasible to lp. Let us split the set of n integer variables into 2 subsets, such that $n = p + q, p \geq 0, q \geq 0$, $s_p \in \mathbb{Z}^p$, $s_q \in \mathbb{Z}^q$ and thus $s = s_p \cup s_q$. The basic principle is that optimizing a linear program $lp' = lp \cap s_p$ generates a feasible solution s' to lp' which is also feasible to lp and at least as good as s. All variables and constraints of the original problem are still present in the new problem, but cuts were added, which limit the solution space based on s_p, resulting in a more restricted solution space. In general, the smaller is the solution space, the easier is the solver convergence.

The aim of cutting the solution space is not to eliminate non-promising search-space regions, but to search in a smaller and treatable space. There is then no guarantee that the optimal solution $s*$ for lp is in or out the eliminated region, but it is ensured that the optimal solution $s'*$ for lp' is equal or better than the previously known solution s. It is thus not interesting to keep these cuts after solving lp'. Instead, the idea is to restore the original problem and add others cuts, now based on the new solution s'. Doing that iteratively, we create a guide for the solver.

So, what solution space of lp' looks like? At each polishing iteration, the limitation of the search-space takes place through the tightening of a subset of variables bounds based on the initial solution s. Suppose the trivial example where the solution space is equal to the cube $c = [0, 4]^3 \in \mathbb{Z}^3$ and the feasible solution $s \in \mathbb{Z}^3$ is $s = \{s_1 = 3, s_2 = 2, s_3 = 0\}$. A possible limitation based on s would be to tighten the interval bounds of the first dimension to the value of s_1, such that the resulting solution space is $c' = [3, 3][0, 4][0, 4]$. Such tightening clearly eliminates one dimension of the search-space and keeps the feasible solution s. Consequently, optimizing over c' can not lead to a worse solution, because in the worst-case scenario the best solution is s itself.

Because at the beginning of the polishing process the solution is supposedly a very poor quality one and the search-space is a wide unexplored space, the percentage of variables with tightened bounds starts high (lets say around 90%) and drops as the limited search-space is potentially exhausted and solution improvement stagnates. If at the end of the polishing this percentage reaches 0%, then the considered linear program is the original one and its optimization ends the process. If an optimal solution is found at the 0% stage, then it is surely an optimal solution for the original problem. For too hard linear programs though, it is unlikely that the polishing reaches this last stage, and the process is then interrupted by time limit. Generally speaking, nothing can be said about optimality for a solution found before the 0% stage.

4.3.2 The polishing algorithm developed for the problem

Following a detailed algorithm for the polishing method developed for the school timetabling problem is described. The main algorithm, specified in Algorithm 4.1, contains the general steps of the polishing process.

The essential requirements for using the polishing method are a linear program and a feasible initial solution. In addition, in case a proved optimal solution is not achieved, time limits are used as stopping criteria: a total maximum execution time for the method ($maxTime$), and a maximum execution time since the last solution improvement occurrence ($maxTimeNoImprov$).

The algorithm starts at line 2 with some initialization, defined at line 11. The current solution sol is set with the initial solution, the percentage $perc$ of variables with fixed bounds is initialized with 90%, the MIP optimization time limit $timeIter$ is initialized with 70 seconds, all blocks are set free and, in the case of several blocks which share teaching resources, blocks may be clustered to assist further bounds tightening. The polishing process begins at line 4, where at each iteration a new optimization cycle takes place. Every cycle has 4 main steps: variables bounds tightening (line 5), optimization of the tightened linear program (line 6), update of percentage of fixed variables and of time limit for each optimization (line 7), and variables bounds release (line 8). At line 9 the polishing time limit is checked.

Algorithm 4.1 Polishing method

Require: *lp* model, any initial feasible solution *solIni*, the maximum runtime *maxTime* for the whole polishing method and the maximum run time *maxTimeNoImprov* of polishing without improvement

Ensure: A final feasible solution *sol* for the *lp* model at least as good as the initial solution *solIni*

```
 1: function POLISH($solIni$, $maxTime$, $maxTimeNoImprov$, $lp$)
 2:     INIT(lp,solIni)
 3:     $okIter \leftarrow true$
 4:     while $okIter$ do
 5:         FIXVARS(lp,sol,perc)
 6:         OPTIMIZE(lp,sol)
 7:         UPDATEPERCANDTIMEITER()
 8:         UNFIXVARS()
 9:         CHECKRUNTIMELIMIT()
10:     return $sol$
```

Initialization

```
11: procedure INIT($lp$, $solIni$)
12:     $sol \leftarrow solIni$
13:     $perc \leftarrow 90$
14:     $timeIter \leftarrow 70$
15:     set all blocks free
16:     cluster blocks according to teaching staff sharing
```

Algorithm 4.2 describes the step of tighten variables ranges. Different approaches were tested and those which best succeed are shown. The main algorithm requires the original linear program, a feasible solution and the percentage of fixation; and ensures a resultant linear program for which a subset of variables have tightened bounds and where the subset size is proportional to the fixation percentage. Three different types of fixation were implemented: fixing all lessons of a subset of blocks (line 2), fixing lessons of random classes (line 3) and fixing random assignments of professors to classes (line 4). The procedure for fixation of lessons per block, detailed at line 5, requires the set FB of blocks that should be completely fixed at the current iteration, besides a feasible solution for the *lp*. Then, every potential lesson that belongs to a fixed block has the lower and upper bounds of its corresponding variable x tightened according to its value at the current solution $sol[x]$. The procedure for fixation of random classes, detailed at line 9, requires a percentage value $perc$ that indicates how much of the current solution should be fixed. Such amount of classes is then randomly selected and stored in the set FC (lines 11- 13). Following, analogously to blocks fixation, every lesson variable x for which the class belongs to FC has its lower and upper bounds tightened according to its value at the current solution $sol[x]$ (lines 14- 16). Lastly, the procedure for fixation of random professors assignments, detailed at line 17, likewise requires the solution fixation percentage, selects the corresponding amount of variables y responsible for assigning professors to classes and tights their bounds.

Algorithm 4.2 Fixing variables' values

Require: *lp* model, a feasible solution *sol* and the percentage *perc* of fixation
Ensure: A resultant *lp* model with a random subset of its variables with fixed bounds and therefore easier to solve

```
1: procedure fixVars($lp$, $sol$, $perc$)
2:     fixBlocks(lp,sol,FB)
3:     fixVarsClassesBounds(lp,sol,perc)
4:     fixVarsProfBounds(lp,sol,perc)
```

Fixing blocks

Require: *lp* model, a feasible solution *sol* and a set FB of blocks to be fixed

```
5: procedure fixBlocks($lp$, $sol$, $FB$)
6:     for each variable $x_u \in lp$ such that $u \in FB$ do
7:         $lowerbound_x \leftarrow sol[x]$
8:         $upperbound_x \leftarrow sol[x]$
```

Decide which classes to fix and fix the corresponding lessons bounds

Require: *lp* model, a feasible solution *sol* and the percentage *perc* of fixation
Ensure: A subset of lessons variables with tightened bounds

```
9: procedure fixVarsClassesBounds($lp$, $sol$, $perc$)
10:     $FC \leftarrow \emptyset$
11:     for each variable $z \in lp$ do
12:         if $rand() \geq perc$ then
13:             $FC \leftarrow$ class represented by $z$
14:     for each variable $x \in lp$ such that class of $x$ is in $FC$ do
15:         $lowerbound_x \leftarrow sol[x]$
16:         $upperbound_x \leftarrow sol[x]$
```

Fix bounds of variables of professors

Require: *lp* model, a feasible solution *sol* and the percentage *perc* of fixation

```
17: procedure fixVarsProfBounds($lp$, $sol$, $perc$)
18:     for each variable $y \in lp$ do
19:         if $rand() \geq perc$ then
20:             $lowerbound_y \leftarrow sol[y]$
21:             $lowerbound_y \leftarrow sol[y]$
```

Algorithm 4.3 describes the steps around a request for optimization. It requires a linear program *lp* and a feasible solution *sol*, and ensures a final updated solution *sol* at least as good as the initial one. The linear program considered here is the one resulting from the variables bounds tightening step. The procedure sets the time limit *timeIter* for the optimization, sets *sol* as an initial solution and optimizes the model using a MIP solver. When optimization ends, run time is recorded at *runtime* and the extra time, i.e., the difference between time limit and run time, is recorded at *timeLeft*. These values may later be used for updating *timeIter* with a more appropriate value. Finally, if optimization succeeded, the current solution *sol* is updated with the new solution and elapsed time since last solution improvement is updated.

Algorithm 4.3 Optimize

```
Require: $lp$ model, a feasible solution $sol$ to $lp$
Ensure: A final updated feasible solution $sol$ for the $lp$ model at least as good
    as the initial solution
 1: procedure OPTIMIZE($lp$, $sol$)
 2:     set optimization time limit equal to $timeIter$
 3:     set $sol$ as a start solution to $lp$
 4:     optimize $lp$
 5:     $runtime \leftarrow$ time spent at the optimization
 6:     $timeLeft \leftarrow |timeIter - runtime|$
 7:     if optimization succeeded then
 8:         $sol \leftarrow$ new best solution found
 9:         if solution was not improved then
10:             update elapsed time without improvement
11:         else
12:             reset elapsed time without improvement
```

Algorithm 4.4 is responsible for taking actions, whether there is a need, on percentage of solution fixation or on optimization time limit. Such actions are based on the last optimization performance. At line 2 it is checked if the linear program is completely free, i.e., if it is the original *lp*. If so, then the current iteration is the last one and there is nothing left to be updated. Otherwise, actions differ accordingly whether the *lp* was solved to optimality (line 5) or not (line 7). If a proven optimal solution is found, it suggests that the *lp* was not hard, and actions may be decrease the portion of the problem that is fixed (line 9) and adjust optimization time limit (line 10). On the other hand, if optimality was not reached, it suggests that the *lp* may be too hard. Thus, in case of solution has not been improved when compared to the previous iteration (line 12), efforts to facilitate the problem are made, such as updating the set of fixed blocks (line 14) and adjusting time limit (line 15).

Algorithm 4.4 Update percentage of fixed variables, fixed blocks and maximum time per iteration

```
Require: lp model, $MaxTimeImposedPerIteration$ and values of configuration and performance for the current iteration optimization: $timeIter$, $runtime$, $timeleft$, $perc$, $percBlockFixed$
 1: procedure updatePercAndTimeIter($lp$)
 2:     if allFree() then
 3:         $okIter \leftarrow false$ return
 4:     if lp was solved until optimal or gap is small enough then
 5:         updatePercAndTimeIterSmallGap(
                $MaxTimeImposedPerIteration$, $timeIter$, $runtime$,
                $timeleft$, $perc$, $percBlockFixed$)
 6:     else
 7:         updatePercAndTimeIterBigGap(
                $MaxTimeImposedPerIteration$, $timeIter$, $runtime$,
                $timeleft$, $perc$, $percBlockFixed$, $percBlockFixed$)
```

Update percentage of fixed variables, fixed blocks and maximum time per iteration for small gap case

```
 8: procedure updatePercAndTimeIterSmallGap(
            $MaxTimeImposedPerIteration$, $timeIter$, $runtime$,
            $timeleft$, $perc$, $percBlockFixed$)
 9:     adjustPercOrBlock($perc$, $timeIter$, $timeLeft$)
10:     adjustTime($MaxTimeImposedPerIteration$, $timeIter$, $runtime$,
    $timeleft$, $perc$, $percBlockFixed$)
```

Update percentage of fixed variables, fixed blocks and maximum time per iteration for big gap case

```
11: procedure updatePercAndTimeIterBigGap(
            $MaxTimeImposedPerIteration$, $timeIter$, $runtime$,
            $timeleft$, $perc$, $percBlockFixed$, $percBlockFixed$)
12:     if solution was not improved at the current iteration then
13:         $acresm \leftarrow 0$, if all blocks are currently free; or 10 otherwise
14:         setNextRandFreeBlocks($acresm$, $percBlockFixed$)
15:         adjustTime($MaxTimeImposedPerIteration$, $timeIter$,
             $runtime$, $timeleft$, $perc$, $percBlockFixed$)
```

Algorithm 4.5 is a procedure to adjust the set of blocks that will be free or fixed at next iteration. It requires an incremental value *percAdjust* to the percentage of fixed blocks, which can be negative if the aim is to hamper the problem, null if no change of percentage is aimed, or positive if the aim to facilitate the problem. Such percentage (*percBlockFixed*) is updated at line 2. At line 3 the number of consecutive iterations with fixed blocks is verified and, if it is sufficient, all blocks are set free for the next iteration. The concept of what is sufficient is subjective, but tests have worked well with a maximum of 4 consecutive iterations. Otherwise, blocks to set free at next iteration are chosen by a routine at line 5. Such routine resets the current set of free blocks, and decides randomly at line 8 whether the type of block fixation will be by cluster or not. If so, a cluster of blocks is selected at line 10 and every block of the cluster is set free. Then, the number of fixed blocks is checked and, while it doesn't reach at least the percentage *percBlockFixed*, blocks are set free randomly.

Algorithm 4.5 Set free and fixed blocks for the next iteration

```
Require: an incremental percAdjust of the fixed blocks percentage
    percBlockFixed
Ensure: an updated set of fixed or free blocks for next iteration

1: procedure setNextRandFreeBlocks(percAdjust, percBlockFixed)
2:     $percBlockFixed \leftarrow percBlockFixed + percAdjust$
3:     if there is enough consecutive iterations with fixed blocks then
4:         free all blocks for the next iteration return
5:     chooseAndSetFreeBlocks(percBlockFixed)
```

Choose and set free blocks for the next iteration

```
6: procedure chooseAndSetFreeBlocks(percBlockFixed)
7:     reset set of free blocks
8:     useCluster ← true or false, randomly chosen with 50% of chance of true

9:     if useCluster then
10:        choose a cluster of blocks and set it free
11:    while number of free blocks < $(100 - percBlockFixed) \cdot$ total of blocks
   do
12:        choose randomly a block and set it free
```

Algorithm 4.6 adjusts optimization time limit for the next iteration according to the current iteration performance. If the linear program was set completely free for next iteration, which means it is the original problem, then it will be the last optimization, and therefore all remaining time for polishing is set. If solution was not improved and time limit was reached before proving optimality, then it suggests that time limit may be too low and its increasing is proceeded at line 5. Otherwise, if solution was improved, then time limit is checked at line 7. The increasing of time, described at line 8, is proportional to the current percentage of fixed variables and blocks. Just as a precaution so that time limit does not increase too much, an upper bound for it may be forced at line 13. The decreasing of time, described at line 15, occurs whether the current time limit has been shown to be much higher than necessary, which is measured by comparing the difference between the run time and the time limit ($timeLeft$) and the extra time required for the sake of precaution ($minExcess$).

Algorithm 4.6 Adjust time

Require: *MaxTimeImposedPerIteration* and values of configuration and performance for the current iteration optimization: *timeIter*, *runtime*, *timeleft*, *perc*, *percBlockFixed*

Ensure: optimization time limit adjusted for next iteration

```
1: procedure adjustTime(($MaxTimeImposedPerIteration$, $timeIter$, $runtime$, $timeleft$, $perc$, $percBlockFixed$))
2:     if allFree() then
3:         $timeIter \leftarrow getRemainingTime()$ return
4:     if solution was not improved at the current iteration and time limit for the iteration was reached then
5:         increaseTime($MaxTimeImposedPerIteration$, $timeIter$, $perc$, $percBlockFixed$)
6:     if solution was improved at the current iteration then
7:         decreaseTime($timeIter$, $runtime$, $timeleft$)
```

Increase time

Require: *MaxTimeImposedPerIteration* and values for the current iteration optimization: *timeIter*, *perc*, *percBlockFixed*

Ensure: updated *timeIter*

```
8: procedure increaseTime(($MaxTimeImposedPerIteration$, $timeIter$, $perc$, $percBlockFixed$))
9:     $incremTime \leftarrow 10$
10:    $incremTime \leftarrow incremTime + (100 - perc) \cdot 0.2$
11:    $incremTime \leftarrow incremTime + (100 - percBlockFixed) \cdot 0.2$
12:    $timeIter \leftarrow timeIter + incremTime$
13:    if $timeIter > MaxTimeImposedPerIteration$ then
14:        $timeIter \leftarrow MaxTimeImposedPerIteration$
```

Decrease time

Require: values for the current iteration optimization: *timeIter*, *runtime*, *timeleft*

Ensure: updated *timeIter*

```
15: procedure decreaseTime(($timeIter$, $runtime$, $timeleft$))
16:    $minExcess \leftarrow max(0.5 \cdot timeIter, 50)$
17:    if $timeLeft > minExcess$ then
18:        $timeIter \leftarrow runtime + minExcess$
```

Algorithm 4.7 is responsible for adjusting the percentage of solution that is fixed. The decrease of fixed solution amount can take place in 2 situations: either optimality was very fast achieved (line 2) or optimality was achieved but solution was not improved at all (line 5). The first one is not really necessary — it is basically just an attempt to speed up the method. The second one is essential, because it is the natural way of decreasing fixation, which means getting closer to the actual linear program. The idea is that the search-space size should be enlarged whenever the optimal solution value equals the current best solution value, since it suggests that search-spaces with the current size may have been exhausted or would make small contributions. The subroutine for decreasing solution fixation, called DECREASEPERCORFREEBLOCK and described at 7, requires a value that might be subtracted of the fixation percentage. It can actually either decide to change and enlarge the set of free blocks (line 9) or indeed subtract the percentage *perc* of solution fixation (line 11). The principle is that decreasing the value *perc* is an "one way street", i.e., we can never backtrack and increase *perc*, while on the other hand the number of fixed blocks can vary according to the difficulty faced when solving *lp*. Thus, the actual decreasing of the value *perc* takes place only when all blocks are free at the current iteration.

Algorithm 4.7 Adjust percentage of fixed variables or fixed blocks

```
Require: values of performance of the current iteration optimization
Ensure: if necessary, updates portion of tightened variable's bounds for the
    next optimization
1: procedure adjustPercOrBlock(($perc$, $timeIter$, $timeLeft$))
2:     if $optimal$ and $timeLeft > 0.7 \cdot timeIter$ then
3:         decreasePercOrFreeBlock$(\text{perc}, 5)$
4:     else
5:         if $optimal$ and not $improved$ then
6:             decreasePercOrFreeBlock$(\text{perc}, 10)$
```

Decrease percentage of fixed variables or free blocks

```
Require: a value $percToSubtract$ that may be subtracted of the fixation
    percentage $perc$
Ensure: updated portion of tightened variable's bounds for the next opti-
    mization, attempting to solve a lp more similar to the original one
7: procedure decreasePercOrFreeBlock($perc$, $percToSubtract$)
8:     if $percToSubtract \leq 0$ then return
9:     if there is any block currently fixed then
10:        setNextRandFreeBlock$(-10)$ return
11:    $perc \leftarrow perc$ - $percToSubtract$
```

Before ending any polishing iteration it is mandatory that every tightening of variable bounds is undone. Algorithm 4.8 is responsible for such untightening, so that the resulting linear program is the same one of the beginning of the iteration.

Algorithm 4.8 Unfixing variables' values

```
Require: $lp$ model
Ensure: updated $lp$ with original bounds for those variables that have been
    tightened
1: procedure unfixVars(($lp$))
2:     for each variable $var \in lp$ that was fixed do
3:         $lowerbound_{var} \leftarrow$ original lower bound of var
4:         $upperbound_{var} \leftarrow$ original upper bound of var
```

The last action at each iteration is to check polishing run time. As previously stated, in case a proven optimal solution is not achieved, two possible stopping criteria are used: total run time and elapsed time since last solution improvement. Both cases are tested in Algorithm 4.9 and, if any of these conditions is reached, *okIter* is set to false, which causes further interruption of the method.

Algorithm 4.9 Check run time

```
Require: Data involving run time and performance
Ensure: Updated $okIter$
 1: procedure CHECKRUNTIMELIMIT($maxTimeNoImprovement$,
            $totalruntime$, $maxTime$, $okIter$)
 2:     CHECKTOTALRUNTIME($totalruntime$, $maxTime$, $okIter$)
 3:     CHECKTIMEWITHOUTIMPROV($maxTimeNoImprovement$, $okIter$)
```

Check total run time

```
Require: $totalruntime$, $maxTime$, $okIter$
Ensure: updated $okIter$
 4: procedure CHECKTOTALRUNTIME(($totalruntime$, $maxTime$, $okIter$))
 5:     if total run time $\geq$ maxTime then
 6:         $okIter \leftarrow false$
```

Check run time since last improvement

```
Require: Information about improvements on the best solution,
    $maxTimeNoImprovement$, $okIter$
Ensure: updated $okIter$
 7: procedure CHECKTIMEWITHOUTIMPROV(($maxTimeNoImprovement$,
    $okIter$))
 8:     if no improvement was made at the current iteration then
 9:         if elapsed time since last improvement > maxTimeNoImprov then
10:             $okIter \leftarrow false$
```

5
Computational Experiments

This chapter describes the main computational experiments conducted in this work. First, general computational aspects are introduced; then different scenarios for different schools are described; and finally results are presented and evaluated.

5.1
Computational resources

The implementation of the solver was done in C++, compiled with MS Visual Studio 2010, using Microsoft Windows 7, 64 bits. All the linear integer programs were solved using the generic MIP-Solver Gurobi 6.0. Computational experiments were executed on 3.40GHz Intel Core i7 computer with 32 GB of RAM. Considering all scenarios that are following described, the maximum amount of RAM used by the solver during execution was around 7 GB.

5.2
Scenarios

Several computational experiments have been performed for real scenarios of Brazilian schools. Those considered the most significant are following described.

The first column of table 5.1 contains some general problem data features, considering the most relevant ones, while the other columns represent all scenarios (A, $B1$, $B2$, C, D, E). First displayed features are the total number of classes, the total number of credits demanded by the classes, and the average number of credits demanded per class. Following there are the total number of teachers and the total number of teaching staff available credits. Lastly, the total number of blocks, that is, distinct school units, the minimum and maximum number of classes per block, and the average number of classes per block are displayed. Notice that consistency of teaching staff availability depends on how it is distributed along the week and on its intersection with demands, courses and students availabilities. The correspondent values exhibited in the table have already been filtered to consider only useful available time slots, i.e., those in such intersection. Daily and weekly availability distributions, though, cannot be extracted from this table.

Table 5.1: General features of scenarios

Features	Scenarios					
	A	B1	B2	C	D	E
Classes	178	295	295	84	24	16
Demanded credits	5800	9490	9490	2709	653	443
Average of credits demanded per class	32.5843	32.1695	32.1695	32.25	27.2083	27.6875
Professors	277	471	471	243	58	35
Teaching staff available credits	9154	15249	15243	7032	900	617
Blocks	14	15	15	4	2	2
Min/Max classes per block	2/29	9/33	9/33	9/28	7/17	2/14
Average of classes per block	15	19	19	21	12	8

As one can see at Table 5.1, the largest scenarios are A, $B1$ and $B2$. They are scenarios of the first semester of 2015 from very large schools of Rio de Janeiro. Scenarios $B1$ and $B2$ correspond to the same school and are then quite similar. Their difference is on the daily availability of professors for teaching at each block — in scenario $B1$ a professor can be assigned to any block where he is supposed to teach on a day that he is available, while in scenario $B2$ an available day of a professor might be previously attached to a smaller subset of blocks, which makes $B2$ more restricted then $B1$. Scenario C was obtained from scenario $B1$ and corresponds to a subset of its blocks. Scenario D refers to the first semester of 2015 from another and much smaller school in Rio de Janeiro. Lastly, E is a scenario of the second semester of 2014 from a school from Minas Gerais.

It is important to say that all considered scenarios have the number of demands per class suitable with the size of the class' available timetable, in the sense that if a class has all its demands satisfied, then its timetable will be compact.

Although the solver is totally prepared to deal with a generic situation, in the clear majority of demands for courses the professor that is potentially going to teach a course section is pre-assigned. In other words, most courses have only one capable professor associated. For example, suppose that Professor Glen Hansard should teach Mathematics to a 2nd Year class in block $BL1$ and Professor Jack Johnson should teach Mathematics to another 2nd Year class in block $BL2$. Then, instead of considering the same `Mathematics` course, we consider two different courses, let us say `Math-2nd-BL1` and `Math-2nd-BL2`, where only Prof. Hansard is capable of teaching `Math-2nd-BL1` and only Prof. Johnson is capable of teaching `Math-2nd-BL2`.

Another relevant feature to have in mind while analyzing a scenario is how courses are organized along the week, that is, the number of credits of a course and its split rule (see section 2.2.2). In scenarios A, $B1$, $B2$ and C most courses have 1 credit each and some courses have 2 credits. In scenarios D and E all courses have 1 credit. Actually, in practice courses have more credits, but for further reasons, which are out the scope, *data was modeled* so that courses with n credits, $n > 1$, were split in n 1-credit courses. It follows that credits split rules were also not necessary.

In order to better portrait dependency between blocks of a scenario, figures 5.2, 5.3, 5.4, 5.5, 5.6 and 5.7 illustrate graphs respectively to scenarios A, $B1$, $B2$, C, D and E, where each node represents a block and each edge represents the sharing of professors between two blocks. Each node has a label in the format "u**X**_**Y**", where **X** is an unique integer identifier for the block and **Y** is the total number of professors that can teach at the block. Every edge has a value that indicates the number of shared professors between the two blocks. Besides, edge's color varies according to how critical to blocks involved is the shared quantity. Figure 5.1 displays the accurate meaning of every used color. A yellow edge indicates low sharing of professors between a pair of blocks, while a black edge indicates very high sharing. The color is not associated with the absolute number of shared professors, but with the percentage of professors shared by the block with less professors. For instance, if block $bl1$ has 10 professors and block $bl2$ has 15 professors, then color of edge $[bl1, bl2]$ is related to $bl1$. If they share 5 professors, then $bl1$and $bl2$ share respectively 50% and 33.3% of their teaching staff, and edge's color is the one associated to the value 50%.

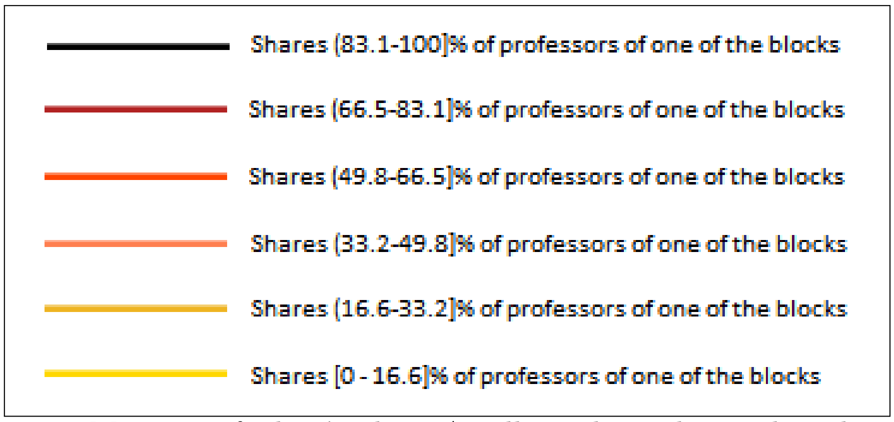
Shares (83.1-100]% of professors of one of the blocks
Shares (66.5-83.1]% of professors of one of the blocks
Shares (49.8-66.5]% of professors of one of the blocks
Shares (33.2-49.8]% of professors of one of the blocks
Shares (16.6-33.2]% of professors of one of the blocks
Shares [0 - 16.6]% of professors of one of the blocks

Figure 5.1: Meaning of edges' colors. A yellow edge indicates low sharing of professors between a pair of blocks and a black edge indicates very high sharing. The color is not associated with the absolute number of shared professors, but with the percentage of professors shared by the block with less professors.

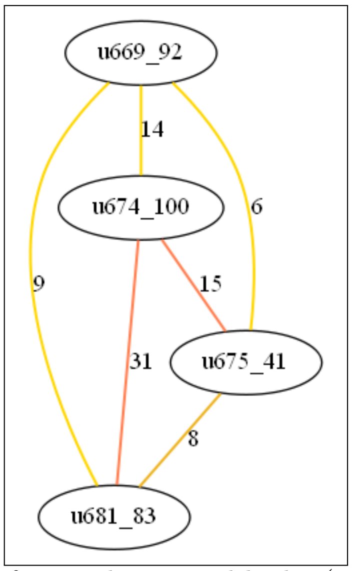

Figure 5.5: Sharing of professors between blocks (school units) of scenario C

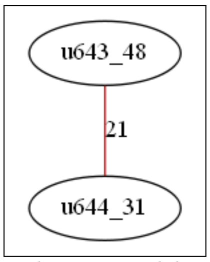

Figure 5.6: Sharing of professors between blocks (school units) of scenario D

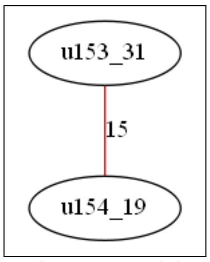

Figure 5.7: Sharing of professors between blocks (school units) of scenario E

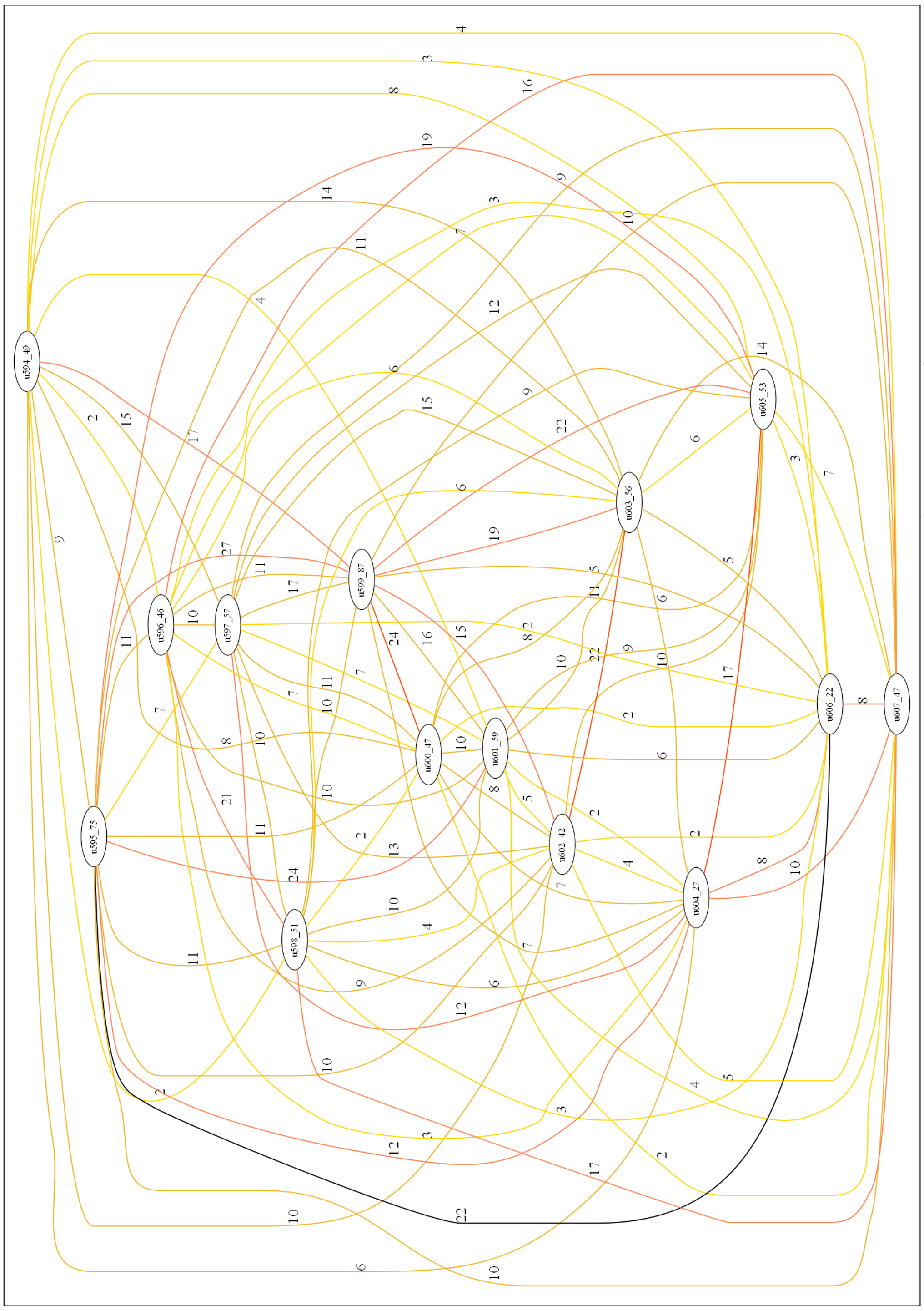

Figure 5.2: Sharing of professors between blocks (school units) of scenario A

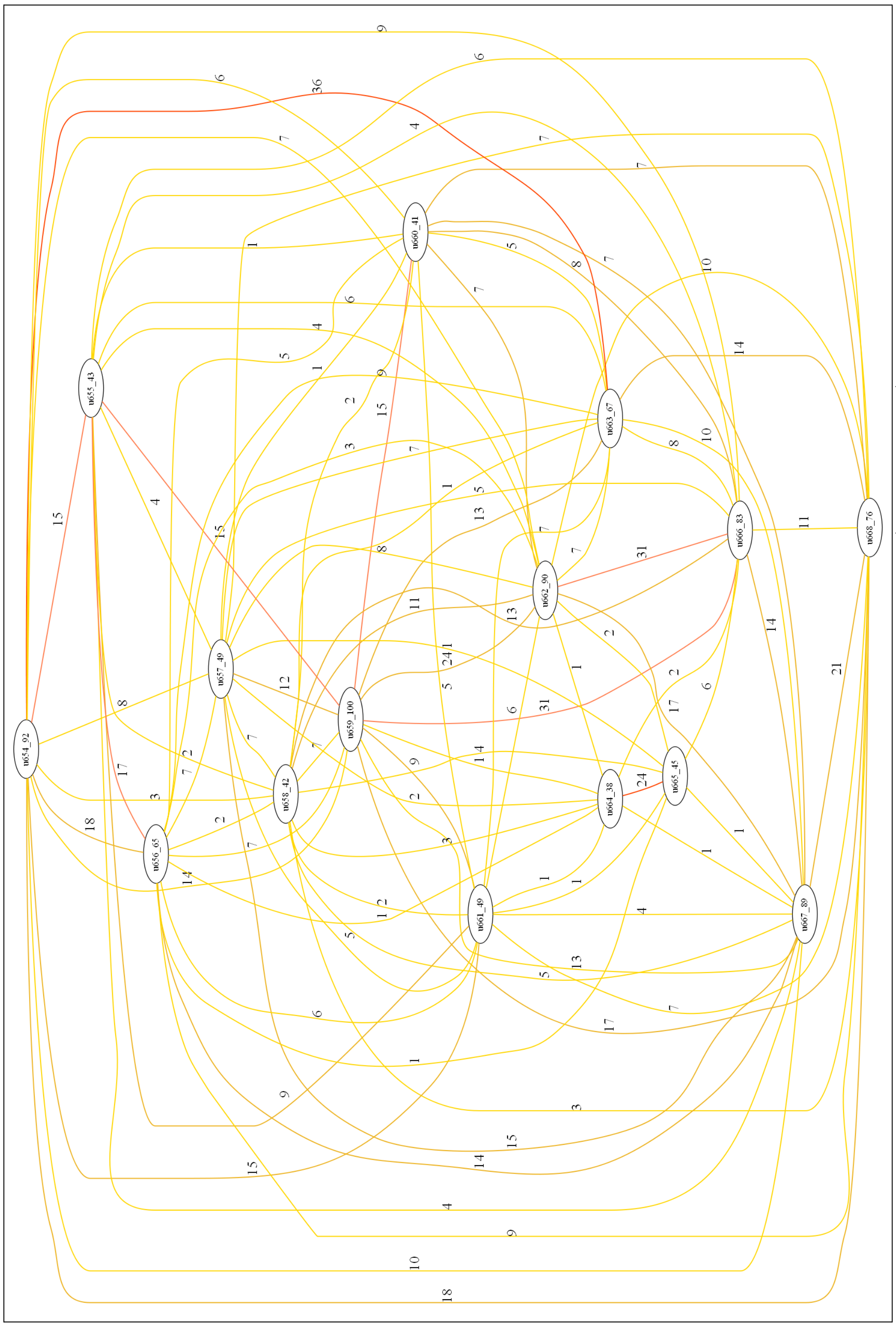

Figure 5.3: Sharing of professors between blocks (school units) of scenario B1

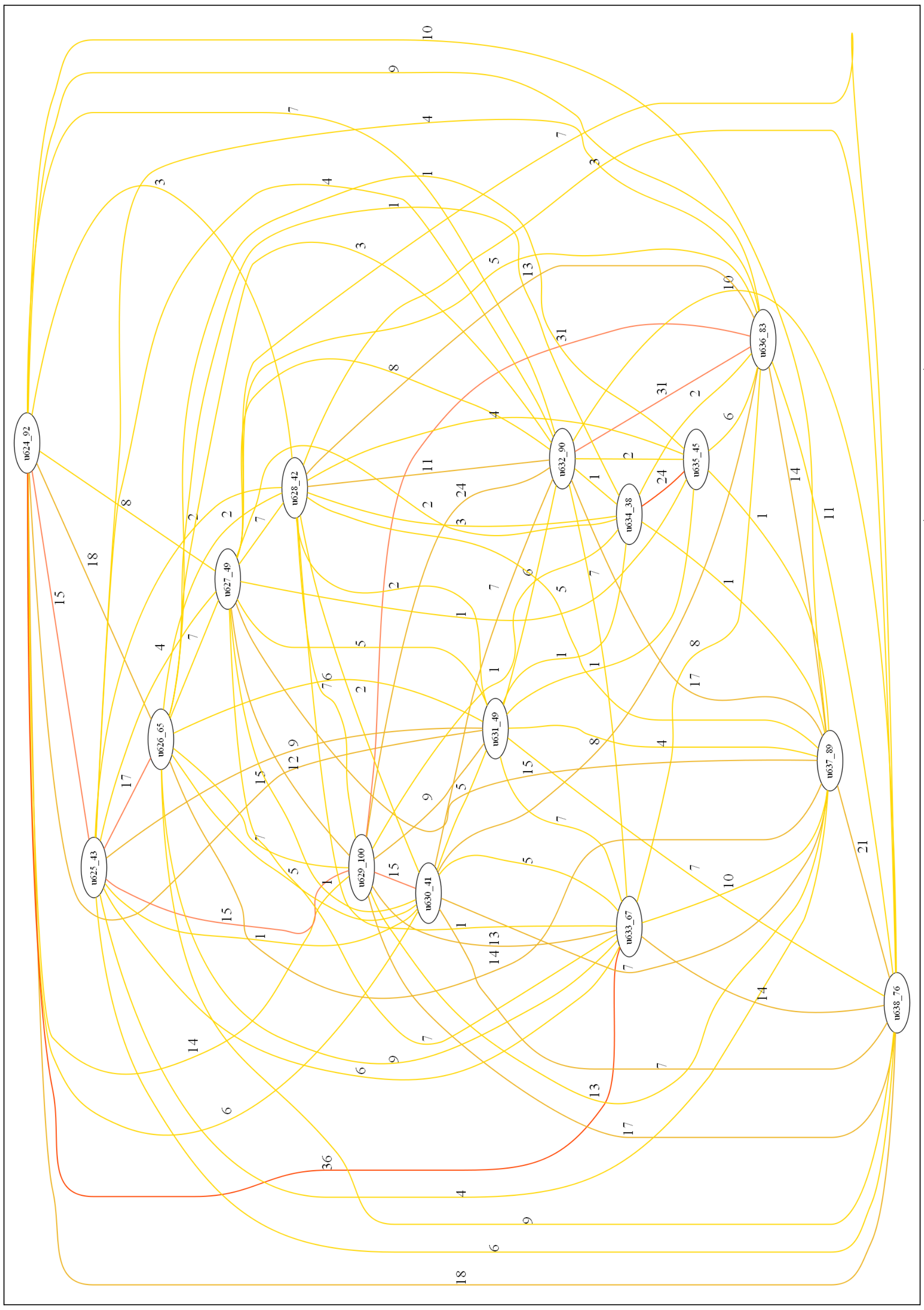

Figure 5.4: Sharing of professors between blocks (school units) of scenario B2

5.3 Approaches

5.3.1 Approaches to combine

Different approaches and strategies for solving the problem were discussed in Chapter 4. Some of these approaches are combined so that further various computational experiments are carried out and impacts of these distinct strategies are evaluated. The variations to be combined are briefly reviewed below.

Real and Virtual Phases Because all considered scenarios have the number of demands per class suitable and coherent with the size of the class available timetable, the experiments were performed without the compactness constraints for students timetables. As explained in Section 4.1, this allows us to split the problem into phases with real and virtual teaching resources, which has shown to be more effective.

Preemptive vs Nonpreemptive Goal Programming The impact of ordered goals, their correlations, and trade-off situations are evaluated by solving the problem both with the preemptive goal programming and with an unified objective function.

Polishing method usage The need of the polishing method is demonstrated by optimizing the problem with and without it.

Professor Priority Professors priorities are evaluated using the three following perspectives:

1. No professor priority is considered: every goal of the problem is executed only once, considering all professors assignments without distinction.
2. Weak professor priority is considered: every goal of the problem is executed only once, but considering first-priority professors assignments more important than second-priority professors.
3. Strong professor priority is considered: every goal of the problem is executed twice. First, all goals are sequentially performed for all first-priority professors, then solution quality that was achieved for these professors is ensured and all goals are again sequentially performed for all second-priority professors.

5.3.2 Gurobi Parameters

Regardless of phase or step, the main configuration used for Gurobi parameters was:

– setting cuts to 2 (aggressive),

– barrier and crossover method for solving root relaxation,

– setting mip focus to 1 (emphasis on finding new solutions).

Besides, in Table 5.2 the number of hours used as time limit in experiments are exhibited. Time limit varies according to the step, in case of using the preemptive goal programming approach, or is a single value in case of using a multi-objective function.

Table 5.2: Time limits per step

Step	Total time limit (h)	Time limit since last solution improvement (h)
No Satisfaction	4	1
Sharable Classes	1.5	0.75
Prof. Displac.	2	1
Prof. Sessions/Days	1	0.5
Prof. Gaps	1	0.5
Multi-objective	4	0.5

5.4 Results

Each scenario was tested combining the different approaches. Due to long runtime, only a few tests were performed using professors priorities. The majority of experiments, which are following detailed, combines either preemptive or nonpreemptive goal programming with either the usage or not of the polishing method, with no professors priorities. Besides, because of the random factor, especially of the polishing method, each pair [scenario+approaches combination] was performed 5 times. Results for all experiments are attached in Appendix B.

5.4.1 Model features

Model features for each instance are listed in Table 5.3, which includes the number of variables and restrictions created, detailed by type, and the number of non-zeros coefficients in matrix. The first column lists the evaluated features and the other columns identify the scenarios. The first lines display general features of the linear program for every scenario, then the number of variables is specified according to each variable type, and analogously the number of constraints is specified according to each constraint type. The number by the side of each constraint type identifies the corresponding equation of the model formulation introduced in section 3.2.2.

Table 5.3: Model features

Lp Features		Scenarios					
		A	B1	B2	C	D	E
general	Total of variables	350132	594528	528315	167276	32225	26198
	Total of constraints	989783	1426770	1113371	293891	45523	32757
	Total of non-zeros	11083636	15918011	11247214	2435153	313373	197020
variables	x	101882	172126	150719	48320	9256	7689
	v	104052	181780	160363	49816	9264	7893
	o	5612	9135	9135	2613	653	449
	s	5934	9723	9723	2725	657	461
	fd	5613	9183	9183	2625	653	443
	z	5612	9135	9135	2613	653	443
	k	102214	173162	151748	48590	9256	7623
	y	5612	9145	9145	2616	653	443
	uu	2174	3278	2904	1055	210	147
	displac	4978	5836	4372	580	128	0
	hip/hfp	2740	5264	5198	2458	328	250
	ptf	1370	2632	2599	1229	164	125
	pt	899	1497	1492	807	146	107
	begin/end	0	0	0	0	0	0
	pfgap	1370	2632	2599	1229	164	125

	LinkVX (3.4)	101882	172126	150719	48320	9256	7689
	RoomUsedTimeSlot (3.7)	7637	13115	13056	3681	754	483
	1LessonDay (3.14)	17051	29872	26248	8366	1820	1487
	SatisfyClass (3.8)	5613	9183	9183	2625	653	443
	LimitClassOffer (3.9)	5612	9135	9135	2613	653	443
	LimitMaxClassSize (3.10)	5612	9135	9135	2613	653	443
	ClassTime (3.13)	8677	17623	17554	4937	754	477
	1RoomCourseSection (3.6)	5612	9135	9135	2613	653	443
	CourseSectionRoom (3.5)	5612	9135	9135	2613	653	449
	ClassCourseLessons (3.11)	5934	9723	9723	2725	657	461
	LinkXZ (3.12)	5612	9135	9135	2613	653	443
	NoOverlapProfTT (3.23)	10146	17269	17051	8085	960	617
	ProfLesson (3.24)	102214	173162	151748	48590	9256	7623
	ProfLessonSum (3.25)	102214	173079	151665	48576	9256	7623
constraints	ProfCourseSection (3.26)	5612	9145	9145	2616	653	443
	CourseSect1Prof (3.27)	5612	9135	9135	2613	653	443
	ProfHiUB (3.36)	9010	15200	14968	7021	897	617
	ProfHfLB (3.37)	9010	15200	14968	7021	897	617
	ProfHiLB (3.35)	7640	12568	12369	5792	733	492
	ProfHfUB (3.38)	7640	12568	12369	5792	733	492
	ProfGapMTN (3.39)	1370	2632	2599	1229	164	125
	ProfDaySession (3.40)	1370	2632	2599	1229	164	125

ProfDay (3.41)	899	1497	1492	807	146	107
DisplacTimeProf (3.28)	50212	71623	50534	5281	633	0
Max1DisplacProf (3.33)	493480	602211	381093	63217	2729	0
BlockProf (3.30)	2174	3278	2904	1055	210	147
SetDisplacProf (3.32)	4978	5836	4372	580	128	0
MaxDispProf (3.33)	571	994	844	206	64	0
MaxDispProfWeek (3.34)	173	293	249	63	21	0
IdleTimeBetSess (3.42)	437	976	954	359	17	16

By observing the number of constraints created for each type of restriction, in Table 5.3, it is possible to know which restrictions were considered in each scenario. In case a variable or constraint type is not displayed at the table, simply assume that it was not created. For example, because all scenarios evaluated have students' availabilities compatible with their demands, all constraints and variables necessary only for avoidance of gaps in students' timetables were not used.

Particularly for scenarios $B1$ and $B2$, which basically differ from each other in the possible assignments between professors and blocks on each day, notice that the biggest differences at the number of constraints created are those related to professor displacement. Besides, constraints of type `Max1DisplacProf` are in some scenarios responsible for almost 50% of rows of the linear program.

5.4.2 Solver performance

Whenever the solver is executed, analyzing performance and consequently the generated solution implies in analyzing every solving step. This means checking for every step the run time, the optimization stopping condition that was reached, the best solution value of the linear program, the optimality gap and also how the solving process converged during the polishing method.

Next are displayed performance details for *some* of the computational experiments that were conducted. Because the purpose here is just to illustrate the process while searching for the best solution, for the sake of simplicity these details are not being exhibited for all experiments.

Table 5.4 refers to scenario B1 considering strong professors priorities, preemptive goal programming and polish approaches, while Table 5.5 refers to scenario A considering professors priorities off, preemptive goal programming and polish approaches. First column of both tables identifies the *solving phase*, while second column lists chronologically all minimization *steps* of the preemptive goal programming approach (see 4.2.2) and the other columns register values for several solver performance features. Phases are split into real and virtual resources phases, while real resource phase can in turn be split into 1st and 2nd-priority in the case of professor priority is considered. Steps are minimization of non-satisfied demands, sharable courses sections, professor displacements, day sessions and days assigned to each professor and gaps in professor timetable. Third column displays the *objective function value* for the best feasible solution found; fourth column displays the *integrality gap* between the best integer solution found and the lower upper bound found at the last polishing iteration (notice that, unless solution fixation percentage is zero at the iteration, this integrality gap is not the one related to the original linear problem); fifth column informs whether the best feasible solution found is proved to be *optimal* (true T or false F); sixth column registers the *final fixed percentage of solution* at the moment when the *polishing* method was interrupted; seventh column displays the *run time* of the step; eighth column displays which *stopping criteria* was reached while solving the step; and lastly ninth column informs the number of *saturated constraints*. Possible stopping criteria for the polishing method are run time limit since last solution improvement (*TNI*), total run time limit (*TL*), zero percentage of fixation plus either time limit or optimal solution (*AF*) and no need of polishing (*NNP*).

Table 5.4: Solver performance for scenario B1 solved using goal programming, polishing approach and strong professor priority

Phase	Step	Feature to minimize						
		OF value	IntGap	Opt	Perc	RunTime	StopCrit	Sat
1st-p	No Satisfaction	34	17.647	F	5	3h39'06"	TNI	864001
	Sharable Classes	6	0	T	0	0h09'02"	AF	864185
	Prof. Displac.	4830	1.035	F	15	2h04'39"	TL	857539
	Prof. Sessions/Days	695	10.36	F	0	0h59'59"	TNI	856923
	Prof. Gaps	300	44.333	F	10	0h56'25"	TNI	856685
2nd-p	No Satisfaction	294	0	F	35	4h01'04"	TL	633637
	Sharable Classes	73	0	T	0	0h46'10"	AF	635671
	Prof. Displac.	25780	0.116	F	30	2h00'50"	TL	633868
	Prof. Sessions/Days	2616	2.332	F	25	1h01'03"	TL	633194
	Prof. Gaps	22837.7	0.64	F	45	1h01'08"	TL	632235
Virtual	No Satisfaction	4.0046	0	T	0	0h0'0"	NNP	40182

Table 5.5: Solver performance for round 1 of scenario A solved using goal programming, polishing approach and professor priority off

Phase	Step	Feature to minimize						
		OF value	IntGap	Opt	Perc	RunTime	StopCrit	Sat
Real	No Satisfaction	221	39.366	F	20	3h28'51"	TNI	525718
	Sharable Classes	50	2	F	10	0h51'34"	TNI	526194
	Prof. Displac.	20680	0.338	F	25	2h04'53"	TL	525697
	Prof. Sessions/Days	1972	0.304	F	25	1h00'43"	TL	525447
	Prof. Gaps	15499.5	4.368	F	35	1h03'15"	TL	525715
Virtual	No Satisfaction	63.0024	0	T	0	0h0'0"	NNP	24537

Similarly, tables 5.6 and 5.7 register the evolution of all goals after each step of the solver. Each line corresponds chronologically to a different solver step focusing on a single goal. Columns 3 to 7 identify all goals of the problem: "Missing Credits" indicates the number of credits that was not satisfied; "Sections" indicates the number of sections used to satisfy demands of sharable courses; "Displac." is the amount of time spent for professors displacements; "Sessions/Days" is the sum of days and days sessions that were assigned to all professors; "ProfGaps" is the amount of idle time in professors timetables, considering the concept of professor gap explained in Section 2.2.2. Columns are ordered according to goals relevance. We draw attention to the fact that for every pair of goals $\{g_1, g_2\}$ where g_1 is more important than g_2, g_1 can never be worsened (in relation to the value achieved at its minimization step) in order to g_2 to get better.

Table 5.6: Goals evaluation for each step for scenario B1 solved using goal programming, polishing approach and strong professor priority

Phase	Step	Feature to minimize				
		Missing Credits	Sections	Displac.	Sessions/Days	ProfGaps
1st-p	No Satisfaction	34	9	14810	730	139789
	Sharable Classes	34	6	11920	730	139789
	Prof. Displacement	33	6	4830	725	139789
	Prof. Sessions/Days	34	6	4830	695	139784
	Prof. Gaps	33	6	4830	695	300
2nd-p	No Satisfaction	294	126	50840	3152	630256
	Sharable Classes	294	73	41850	3133	630256
	Prof. Displacement	294	73	25780	3099	630256
	Prof. Sessions/Days	294	73	25760	2616	630239
	Prof. Gaps	294	73	25700	2616	22837.7
Virtual	No Satisfaction	4	80	34310	3311	518148

In many situations there is a trade-off between the different goals. For example, minimizing professor's teaching days could cause an increase of gaps in professor's timetable. By using preemptive goal programming, this issue is eliminated. But that is definitely not a rule — the opposite can also happen. Trying to minimize professors displacements generally implies more compact timetables, which can result in more possibilities of satisfying demands. Such "mutual cooperation" can be easily verified in Table 5.6, where, for example, minimizing professors displacements generated a reduction of 1 non-satisfied credit in phase 1st-p.

Table 5.7: Goals evaluation for each step for round 1 of scenario A solved using goal programming, polishing approach and professor priority off

Phase	Step	Feature to minimize				
		Missing Credits	Sections	Displac.	Sessions/Days	ProfGaps
Real	No Satisfaction	221	75	32220	2193	441413
	Sharable Classes	221	50	31140	2189	441413
	Prof. Displacement	221	50	20680	2177	441413
	Prof. Sessions/Days	221	50	20670	1972	441392
	Prof. Gaps	221	50	20650	1972	15499.5
Virtual	No Satisfaction	63	51	21390	1972	318101

By evaluating the solver performance when using goal programming, it is possible to see clearly how *symmetric* is the problem. Symmetry in mathematical programming is when two distinct feasible solutions are equality good (or bad), that is, have the same objective function value. For example, in Table 5.7 we see that there are at least 5 different solutions for scenario A with 221 non-satisfied credits, since for every step of the real resource phase some specific goal was improved. Symmetry is reduced whenever a new goal is introduced, which generally also means refinement of solution quality. On the other hand, introducing a new goal increases run time whether solving the problem with preemptive goal programming (because a new step is added) and disturbs the objective function whether solving the problem with a multi-objective function.

5.4.3 Solution quality

Evaluating quality of a final solution is not a trivial task, specially because there are multiple and conflicting goals involved. Often analyzing and understanding solutions require a deeper analysis of the problem data itself. Following several solution quality indicators are listed and detailed for the conducted experiments.

Each table, from 5.8 to 5.13, corresponds to a different scenario and contains the **average** values calculated from results of the 5 rounds performed per approaches combination (results for all experiments can be found in Appendix B). The first column contains the solution features that are going to be evaluated, while the other columns represent different approaches used to solve the scenario. For identifying the approaches, GP and NGP mean respectively preemptive and non-preemptive goal programming, $Pol1$ and $Pol0$ mean usage and no usage of the polishing method, and $PP0$ means professors priorities off. The first displayed feature is the total number of idle time slots in professors timetables that are classified as gaps (see section 2.2.2);

then similarly the total number of idle time slots in professors timetables classified as gaps, ignoring those gaps which include any professor displacement between blocks; and the total number of displacements between *distant* blocks in professors timetables. Following, the total number of professors assigned to classes is displayed and also divided between real and virtual professors. Analogously, the total number of credits assigned to professors are informed with their respective division between real and virtual teaching resource. Then the total number of satisfied courses sections is exhibited, also specifying the real and virtual sections. Lastly are displayed the percentage of credits that was satisfied, the number of satisfied students-demand, and the number of non-satisfied students-demand.

Table 5.8: Average solution quality for scenario A solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	350.4	0.2	384.2	239.2
Prof Gaps ign. displac	184	0.2	230.6	231.6
Prof long displac	53.4	0	43.2	1
Used professors	376.8	94.8	374.8	96.2
Real professors	277	88	277	89
Virtual professors	99.8	6.8	97.8	7.2
Credits assigned to prof	5584.8	453.6	5595	272
↪ to real prof	5408.2	431.6	5404.8	254.6
↪ to virtual prof	176.6	22	190.2	17.4
Courses sections	5467.4	446.8	5470	266
↪ with real prof	5292.2	424.8	5281.2	248.6
↪ with virtual prof	175.2	22	188.8	17.4
% of satisfied credits	98.972	7.82	98.724	4.688
Satisfied stud-dem	5554	446.8	5539.6	266
Non-satisfied stud-dem	59	5166.2	73.4	5347
Total Run Time	9h25'06"	10h12'13"	4h00'35"	8h17'57"

Just a brief glimpse at Table 5.8 is enough to conclude that the usage of the polishing method is essential. For experiments with preemptive goal programming (columns GP-Pol1-PP0 and GP-Pol0-PP0), 98.972% of demanded credits were satisfied when using the polishing method against 7.82% of satisfied demanded credits without such method, with this last approach taking even longer then the first. On the other hand, comparison of the usage of preemptive (GP) and non-preemptive goal programming (NGP) is not so obvious. What is clear and actually expected, based on values introduced by Table 5.2, is that GP usually takes longer than NGP. Solutions obtained with GP are better than with NGP, both in terms of satisfied credits, of number of course sections, of credits assigned to virtual teaching staff and of gaps in real professors timetables, at the expense of longer run time.

Table 5.9: Average solution quality for scenario B1 solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	509	0	476	0
Prof Gaps ign. displac	360.2	0	341.8	0
Prof long displac	127.6	0	104.6	0
Used professors	663.8	0	680.2	0
Real professors	471	0	471	0
Virtual professors	192.8	0	209.2	0
Credits assigned to prof	9268.6	0	9310.4	0
↪ to real prof	8862.2	0	8852.4	0
↪ to virtual prof	406.4	0	458	0
Courses sections	9027.6	0	9062.4	0
↪ with real prof	8625.6	0	8608.4	0
↪ with virtual prof	402	0	454	0
% of satisfied credits	99.978	0	99.978	0
Satisfied stud-dem	9181.8	0	9181.8	0
Non-satisfied stud-dem	1.2	9183	1.2	9183
Total Run Time	10h08'59"	9h07'20"	5h04'23"	8h35'50"

Again the polishing method was essential for scenario B1. We see in Table 5.9 that the satisfied demand was the same in GP-Pol1-PP0 and NGP-Pol1-PP0, but the second one resulted in more courses sections (that is, more credits to be paid to professors, which is cost to the institution) and more credits assigned to virtual professors. Approach GP-Pol1-PP0, in turn, spent twice the run time of NGP-Pol1-PP0, which might not be a worth paying price. Besides, solution of NGP-Pol1-PP0 has less professors long displacements.

Table 5.10: Average solution quality for scenario B2 solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	440.4	0	502.8	0
Prof Gaps ign. displac	341.4	0	388.8	0
Prof long displac	83.2	0	77.8	0
Used professors	638.6	0	653	0
Real professors	471	0	469.8	0
Virtual professors	167.6	0	183.2	0
Credits assigned to prof	9155.6	0	9199.4	0
↪ to real prof	8843.4	0	8828.6	0
↪ to virtual prof	312.2	0	370.8	0
Courses sections	8914.6	0	8950	0
↪ with real prof	8607.6	0	8584.2	0
↪ with virtual prof	307	0	365.8	0
% of satisfied credits	98.792	0	98.804	0
Satisfied stud-dem	9069.4	0	9070.4	0
Non-satisfied stud-dem	113.6	9183	112.6	9183
Total Run Time	9h29'41"	8h41'38"	4h52'51"	8h25'15"

Since scenarios B1 and B2 are just variations of the same actual problem, results in Table 5.9 are quite similar to results in Table 5.10. For scenario B2, solution quality for GP-Pol1-PP0 is better than for NGP-Pol1-PP0. Although satisfied demand was a little bit higher in NGP-Pol1-PP0, it used considerably more credits assigned to virtual professors, with the difference higher than the difference of satisfied demand. On the other hand, GP-Pol1-PP0 took almost double the execution time.

We see that results of scenario B1 had more credits assigned to real professors than B2. This is expected when we think that B1 is more flexible in terms of professors availabilities per pair {day, block} than B2, even though sometimes an increase in feasible solution space can also render it harder to search for good solutions.

Table 5.11: Average solution quality for scenario C solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	97.2	0	205.8	0
Prof Gaps ign. displac	86.8	0	192.4	0
Prof long displac	14.6	0	13.2	0
Used professors	260.2	0	258.8	0
Real professors	243	0	243	0
Virtual professors	17.2	0	15.8	0
Credits assigned to prof	2646	0	2657	0
↪ to real prof	2620.2	0	2630.6	0
↪ to virtual prof	25.8	0	26.4	0
Courses sections	2584	0	2591.6	0
↪ with real prof	2558.2	0	2565.4	0
↪ with virtual prof	25.8	0	26.2	0
% of satisfied credits	99.93	0	99.93	0
Satisfied stud-dem	2624	0	2624	0
Non-satisfied stud-dem	1	2625	1	2625
Total Run Time	5h47’31”	8h07’59”	2h23’09”	8h08’21”

Analysis of solutions for scenario C is similar to the one for B1. Again the polishing method was essential. We see at Table 5.11 that the satisfied demand was the same in GP-Pol1-PP0 and NGP-Pol1-PP0, but the second one resulted in more credits assigned to virtual professor, more courses sections and much more gaps in real professors timetables. The number of professors long displacements was higher in GP-Pol1-PP0, but it was possibly unavoidable, since it uses less courses sections to satisfy the same demand. Approach NGP-Pol1-PP0, in turn, spent less than half the run time of GP-Pol1-PP0.

Table 5.12: Average solution quality for scenario D solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	22.4	42.8	27	36.4
Prof Gaps ign. displac	16.6	38.4	23.6	30.4
Prof long displac	0	0	0	0
Used professors	69.4	68.2	67.2	68.8
Real professors	58	58	58	57.8
Virtual professors	11.4	10.2	9.2	11
Credits assigned to prof	648.2	648.8	648.6	649.4
↪ to real prof	629.8	630.6	631	626
↪ to virtual prof	18.4	18.2	17.6	23.4
Courses sections	648.2	648.8	648.6	649.4
↪ with real prof	629.8	630.6	631	626
↪ with virtual prof	18.4	18.2	17.6	23.4
% of satisfied credits	99.54	99.66	99.63	99.752
Satisfied stud-dem	650	650.8	650.6	651.4
Non-satisfied stud-dem	3	2.2	2.4	1.6
Total Run Time	4h42'21"	4h38'26"	1h07'58"	2h14'24"

Now, for the first time, the polishing method was not that essential. We see at Table 5.12 that satisfied demand was almost the same for all approaches and the number of credits assigned to virtual professor was significantly different only in NGP-Pol0-PP0. Although the highest satisfied demand was in NGP-Pol0-PP0, it is clear that such satisfaction was achieved by using virtual professors, since 23.4 credits were assigned to virtual professors against 17.6, 18.2 and 18.4 credits of the others approaches. Also, in terms of gaps in professors timetables it had a bad result, even using less credits assigned to real professors. The worst approach was then undoubtedly NGP-Pol0-PP0. Comparison of solutions of the others approaches is not so obvious. Still, it seems that NGP-Pol1-PP0 had the best performance. Despite having satisfied a little bit less credits than GP-Pol0-PP0, it used the lowest number of credits assigned to virtual professors, reasonable number of gaps in professors timetables and had by far the lowest execution time. Average solution of GP-Pol1-PP0 had less gaps in professors timetables, but that is expected, since it had also less credits assigned to real professors. Average solution of GP-Pol0-PP0 had the greatest number of satisfied credits and, possibly as a consequence, assigned a little bit more credits to virtual professors, but the number of time slots in gaps in professors timetables was much higher than expected and it took four times longer than NGP-Pol1-PP0.

Table 5.13: Average solution quality for scenario E solved with different approaches

Features	Approaches			
	GP-Pol1-PP0	GP-Pol0-PP0	NGP-Pol1-PP0	NGP-Pol0-PP0
Prof Gaps	0	0	0.4	9.8
Prof Gaps ign. displac	0	0	0.4	9
Prof long displac	0	0	0	0
Used professors	35	35	35	35
Real professors	35	35	35	35
Virtual professors	0	0	0	0
Credits assigned to prof	434	434	434.6	435
↪ to real prof	434	434	434.6	435
↪ to virtual prof	0	0	0	0
Courses sections	434	434	434.6	435
↪ with real prof	434	434	434.6	435
↪ with virtual prof	0	0	0	0
% of satisfied credits	100	100	100	100
Satisfied stud-dem	443	443	443	443
Non-satisfied stud-dem	0	0	0	0
Total Run Time	0h34'48"	0h39'51"	1h09'22"	1h41'28"

For scenario E, the smallest one, we see in Table 5.13 that approaches GP-Pol1-PP0 and GP-Pol0-PP0 are equality good in terms of solution quality, but for the first time GP-Pol1-PP0 was faster. Approach NGP-Pol1-PP0 found a solution slightly worse and took a bit longer (twice the time spent by GP-Pol1-PP0), while approach NGP-Pol0-PP0 was again the worst one, achieving a solution with more credits to be paid by the institution and more gaps in professors timetables, and taking much longer than the others.

Experiments have shown clearly that when size of scenarios increases, the polishing method becomes fundamental for finding a good solution (actually, even any solution). For smaller scenarios the solver was able, without the polishing, to find reasonable solutions, but still in these cases the polishing has also proved to work better than the straight optimization.

In contrast, results for preemptive and non-preemptive goal programming are more questionable. The preemptive goal programming (GP) has the advantage of having a clean objective function at each step, which helps convergence while searching for good solutions. We can easily verify it in almost all experiments exhibited — compare GP-Pol0-PP0 and NGP-Pol0-PP0 for scenario E, or GP-Pol0-PP0 and NGP-Pol0-PP0 for scenario D, or GP-Pol1-PP0 and NGP-Pol1-PP0 for scenario C, and so on. For the smaller scenario, E, the best approach was to use both the polishing method and GP, while for scenario D the best was to use the polishing method with NGP. For bigger scenarios, although GP has generated better solutions than NGP, the difference was not big, while the run time was most times much bigger.

What it is clear is that both the polishing method and the preemptive goal programming assists somehow the MIP-solver in the search for the best solution. Using them simultaneously is not always the best option, but the usage of at least one of them was in all cases advantageous.

6
Conclusions

This dissertation was about timetabling in the school environment. In the school timetabling problem, students are previously grouped into classes and each class is to be assigned to a set of courses. Every course must in turn have its lessons scheduled over the week and assigned to a classroom and a professor. For an actual and applied purpose, besides mandatory operational constraints, such as avoiding resources overbooking, it is essential to consider several solution quality constraints, which can be related to institutional, pedagogical and personal needs.

The most relevant knowledge acquired through experiences lived during development of the Brazilian commercial timetabling software TRIEDA were registered. The system is based on a "demand-drive" philosophy where students previously ask for courses, and having knowledge of the complete institution structure and available resources, the aim is to provide a complete and feasible solution that maximizes the number of satisfied requests while respecting a large set of didactic-pedagogical requirements. For the school environment, some didactic-pedagogical requirements together with professors satisfaction are usually the most important issues.

Because TRIEDA is a commercial software, it has been developed to be as flexible as possible. Furthermore, this gives this work a very practical aspect, in contrast to some too theoretical researches.

For practical timetabling, obtaining coherent and correct data is a particular important and sensitive issue. It is extremely important that data collection phase of an automating process has a deep involvement of all of those who usually operate in the traditional manual process. Any inaccuracy of data can result in a bad or even non-deployable solution.

Automating people assignments is different from automating other process — the optimization factor is definitely **not** more important than people satisfaction. Only keeping this in mind it is possible to obtain an actual deployable solution. Practical course timetabling is actually more politics than graph theory. Professors' availabilities and preferences issue should not be underestimated. The primary design goal is not necessarily finding a true optimal solution, but to assist academic timetablers with the problem of building a better timetable in an efficient way.

An integer linear programming formulation together with some strategies for helping the convergence of the searching process on the best solution were developed for solving the problem. Several computational experiments were conducted using real scenarios of four Brazilian high schools, so that the different approaches could be evaluated.

6.1 Best approaches

Next, general observations and conclusions regarding to behavior and results of the different strategies used for solving the Brazilian school timetabling problem are compiled.

6.1.1 Goal Programming

With regard to the objective function, an alternative approach was to use *preemptive goal programming* (GP), where there is a priority line for the different goals. According to the institution preferences, one optimizes the problem in a sequence of steps, where each step is responsible for a goal, from the most to the least important. For each step a different and specific objective function is used and the feasible solution space is subject to features of the best solution found at the previous step. In contrast, a traditional approach using *nonpreemptive goal programming* (NGP) (multi-objective function) was also tested and results compared.

In many situations there is a trade-off between the different goals. For example, minimizing professor's teaching days could cause an increase of gaps in professor's timetable. By using preemptive goal programming, this issue is eliminated. But that is definitely not a rule — the opposite can also happen. Trying to minimize professors displacements generally implies more compact timetables, which can result in more possibilities of satisfying demands.

Results for GP and NGP were questionable. The preemptive goal programming has the advantage of having a clean objective function at each step, which helps convergence while searching for good solutions, and of not depending on subjective choice of weights for variables in the objective function. For small scenarios, specially without the polishing method, GP has shown to make a great difference if compared to NGP. For larger scenarios, although GP has generated better solutions than NGP, the difference was not big, while the run time was most times much bigger.

Thus, considering problems to which there is a clear priority line, conclusion was that, when run time is not an issue, preemptive goal programming is more reliable and robust. On the other hand, if losing a little bit of solution quality is a worth paying price for reducing run time, then especially for big scenarios a nonpreemptive goal programming may be recommended.

6.1.2 Phases

In case of allowing usage of virtual resources and/or considering professors priorities, a split of the problem into some phases was proposed.

The first division concerned to virtual resources. Originally, allocation of real and virtual resources was decided simultaneously in the same model, by assigning a high penalty for using a virtual resource (in case of multi-objective function) or by adding a new step to minimize virtual resource usage just after maximization of satisfied demand (in case of preemptive goal programming). Such initial approach was necessary due to constraints for preventing idle time slots in students timetables, as explained in Section 4.1. From the moment these constraints are no longer necessary, because students demands fit perfectly their available timetables, we are free to consider virtual resources for those non-satisfied demands only after all real resources were allocated. This partition of the problem led to better results, both in terms of solution quality and of run time. Real resource phase became leaner and meaner, while virtual resource phase is an easy problem and, although it is still solved by an integer linear program, could even be solved by an post-processing algorithm.

The second division referred to professors priorities and actually only makes sense to be used whether the real and virtual resources division is active. Considering a line of reasoning similar to the preemptive goal programming one, if 1st-priority professors strictly precede 2nd-priority professors, then another split can be performed to ensure that assignments for 2nd-priority professors never disturb quality of assignments for 1st-priority professors. Undoubtedly, experiments have shown that the drawback of this approach, especially for big scenarios when merged with the preemptive goal programming, is a significantly increase of run time, which was expected though, since the solver basically runs twice. In case run time is an issue and/or there is not a strict precedence of professors priorities, considering professors priorities simply by differentiating their weights in objective function may be more advantageous.

6.1.3 Polishing method

Regarding the polishing method, experiments have shown clearly that as the size of scenarios increases, the polishing becomes fundamental for finding a good solution (actually, even any solution). For large scenarios without the polishing, Gurobi MIP-solver either could not converge to any integer feasible solution in Branch-and-Bound phase or after hours reached time limit even before finishing root relaxation. For smaller scenarios the solver was able, without the polishing, to find reasonable solutions, but still in these cases the polishing has also proved to work better than the “straight” optimization.

6.1.4
Hard vs soft constraints

Especially by evaluating the solver performance when using goal programming it was possible to see clearly how *symmetric* is the problem. Symmetry is reduced whenever a new goal is introduced, which generally also means refinement of solution quality. On the other hand, introducing a new goal increases run time whether solving the problem with preemptive goal programming (because a new step is added) and disturbs the objective function whether solving the problem with a multi-objective function. It follows that, whenever some potential goal is not too restrictive, it may worth to add it to the formulation as a hard constraint instead as a goal in the objective function with its corresponding soft constraint. This has proved to be true especially for large scenarios, where solution space is huge.

Some constraints proposed in this work concerning to compactness of professors timetables were firstly introduced after experiments have produced solutions with poor quality regarding to displacement of professors between blocks and to distribution of their assignments along the week. Originally they were considered as soft constraints, because in fact their violation still produces, generally, operable solutions. Disturbing the objective function though have proved to be more damaging and less efficient than introducing new hard constraints. Examples of such constraints are equations 3.33, 3.31, 3.33, 3.34 and 3.42.

6.1.5
Fixing partial solution over phases

Another relevant observation concerns to which is the best way to ensure the achieved quality of partial solutions as different phases and steps evolve. Suppose the goal $Minimize \sum_i c_i \cdot x_i$ subject to a set of constraints. After optimizing it and obtaining a solution $x*$, where $g = \sum_i c_i \cdot x*_i$, there are two intuitive ways to guarantee that solutions of next eventual steps do not get a value worse than g for this particular goal:

1. to add a new constraint $\sum_i c_i \cdot x_i \leq g$, or

2. to tighten bounds of all variables involved with this goal according to their values in $x*$.

Adding a new constraint offers the advantage of not eliminating solutions that are symmetric to $x*$ and which could be better than $x*$ concerning to further goals. On the other hand, fixing part of the current solution by tightening bounds of variables involved in the goal can make the next optimization steps easier to solve, since solution space is smaller than in the first case. This leads us to an issue similar to the "Hard vs soft constraints" one, discussed previously. Is tightening of variables bounds too restrictive? Of course it depends on how much these variables define the whole problem. Still, after some experiments, we again realized that in some situations fixing partial solutions helps optimization convergence on next steps more than restricts the solution space. Fixation does not necessarily define *how* an assignment is made, but just that it exists.

In the end, the first option was used to ensure solution quality between steps (minimizing non-satisfied demand, minimizing professor displacement, etc), and the second option was used to ensure solution quality between phases (1st and 2nd-priority professors phases and virtual resources phases).

6.2 Future Work

There is a common situation in schools that was not handle at this work. It is possible that for some few subjects a class is split. Usually it is the case of language courses, where the student have to choose for a foreign language among some options. In this work we handle situations where different classes are merged, but the opposite case is still to be done.

Also, for those schools which provided the test scenarios, credits split rules for courses were not important, that is, courses did not have a particular way to distribute their credits over the week. Usually this is an important requirement for higher education courses, but not for high schools. Consequently, credits split rules were not used in computational experiments documented here.

Finally, avoidance of gaps in student's timetable was obtained implicitly, as explained in Section 4.1, thanks to a fair correspondence between student demands and its available timetable. In case this correspondence fails, partition of the problem into real and virtual resources phases can compromise real resource assignments. On the other hand, deciding simultaneously virtual and real resources assignments makes the problem considerably more difficult and brings us new challenges.

7 Bibliography

ASC Timetables. [S.l.]. Available at `http://www.asctimetables.com/`. 1.6

ASRATIAN, A. S.; WERRA, D. A generalized class-teacher model for some timetabling problems. **European journal of operational research**, vol. 143, p. 531–542, 2002. 1.1

BENCHMARKING project for (high) school timetabling. [S.l.]. Available at `http://www.utwente.nl/ctit/hstt/`. A, A.1

BENLI, O. S.; BOTSALI, A. R. **Decision Support System for Scheduling Courses at Bilkent University**. [S.l.], 2004. Available at `http://www.csulb.edu/~obenli/DSS/info.html`. 1.3

BIRBAS, T.; DASKALAKI, S.; HOUSOS, E. Timetabling for greek high schools. **Journal of the Operational Research Society**, vol. 48, p. 1191–1200, 1997. 1.1, 1.4

BIRBAS, T.; DASKALAKI, S.; HOUSOS, E. School timetabling for quality student and teacher schedules. Kluwer Academic Publishers, vol. 12, p. 177–197, 2009. 1.1, 1.4

CARTER, M. W. A comprehensive course timetabling and student scheduling system at the university of waterloo. In BURKE, E.; ERBEN, W. (Ed.). **Practice and Theory of Automated Timetabling III**. [S.l.]: Springer Verlag LNCS, 2001. p. 64–82. 1.1, 1.3, 1.5, 2.1.1

CURRICULUM-BASED Course Timetabling Project. [S.l.]. Available at `http://tabu.diegm.uniud.it/ctt/`. 1.6

DASKALAKI, S.; BIRBAS, T. Efficient solutions for a university timetabling problem through integer programming. **European journal of operational research**, vol. 160, p. 106–120, 2005. 1.1

DOCUMENTATION for constraints in XHSTT format. [S.l.]. Available at `http://sydney.edu.au/engineering/it/~jeff/hseval.cgi?op=spec&part=constraints`. A.1.5

DOCUMENTATION for XHSTT format. [S.l.]. Available at `http://sydney.edu.au/engineering/it/~jeff/hseval.cgi?op=spec`. A.1

EVENT MAP. [S.l.], 2014. Available at http://www.eventmap-uk.com/. 1.6

GUENALAY, Y.; SAHIN, T. A decision support system for the university timetabling problem with instructor preferences. **Asian Journal of Information Technology**, vol. 12, p. 1479–1484, 2006. 1.1, 1.3, 1.5, 4.2.1

GUROBI. **Gurobi Optimizer**. [S.l.]. Available at http://www.gurobi.com/. 3.1.5

ILOG, I. **IBM ILOG CPLEX**. [S.l.]. Available at http://www-03.ibm.com/software/products/en/category/decision-optimization. 3.1.5

INTERNATIONAL Timetabling Competition (ITC). [S.l.], 2007. Available at http://www.cs.qub.ac.uk/itc2007/. 1.6

INTERNATIONAL Timetabling Competition (ITC). [S.l.], 2011. Available at http://www.utwente.nl/ctit/hstt/itc2011/welcome/. 1.6

KASSICIEH, S. K.; BURLESON, D. K.; LIEVANO, R. J. **Design and Implementation of a Decision Support System for Academic Scheduling**. [S.l.], 1986. vol. 11, 57-64 p. 1.3

LLC, U. **University Timetabling: Comprehensive Academic Scheduling Solutions**. [S.l.], 2007 – 2014. Available at http://www.unitime.org/. 1.3, 1.6

MARTE, M. Models and algorithms for school timetabling – a constraint-programming approach. 2002. 1.1

MIMOSA Scheduling Software. [S.l.], 2014. Available at http://www.mimosasoftware.com/. 1.6

MISTA. **Multidisciplinary international scheduling conference: Theory and Applications**. [S.l.]. Available at http://www.schedulingconference.org/. 1.6

MURRAY, K.; MUELLER, T.; RUDOVA, H. Modeling and solution of a complex university course timetabling problem. In BURKE, E. K.; RUDOVA, H. (Ed.). **Practice and Theory of Automated Timetabling VI**. [S.l.]: Springer Berlin Heidelberg, 2007. p. 189–209. 1.1, 1.3, 1.5

PAPADIMITRIOU, C. H.; STEIGLITZ, K. Combinatorial optimization - algorithms and complexity. Dover, 1998. 3, 3.1.4

PILLAY, N. An overview of school timetabling research. Patat, 2010. 1.1

PRATICE and Theory on Automated Timetabling. [S.l.]. Available at http://www.patatconference.org. 1.6

SANTOS, H. G. et al. Strong bounds with cut and column generation for class-teacher timetabling. Annals of Operations Research, vol. 194, p. 399–412, 2012. 1.1, 1.4, 3

SCHAERF, A. A survey of automated timetabling. vol. 13, p. 87–127, 1999. 1.1

TRIEDA. [S.l.]. Available at http://www.trieda.com.br/. 1.6

UDINE, I. Research Group at the University of. **Research Group on Scheduling and Timetabling**. [S.l.]. Available at http://satt.diegm.uniud.it/home/. 1.6

UNKNOWN. **Chapter 13 - Nonlinear Models: Dynamic, Goal, and Nonlinear Programming**. [S.l.]. Available at http://www.ams.jhu.edu/~castello/625.414/Handouts/GoalProg.pdf. 4.2.1

VALOUXIS, C. et al. Decomposing the high school timetable problem. PATAT, 2012. 1.1, 1.4

(WATT), W. group on automated timetabling. **Watt – Educational Timetabling**. [S.l.]. Available at http://watt.cs.kuleuven.be/. 1.6

WIKIPEDIA. **Education in Brazil**. [S.l.]. Available at http://en.wikipedia.org/wiki/Education_in_Brazil. 1.2

WOLSEY, L. A. Integer programming. Wiley-Interscience, 1998. 3, 3.1.4

A
Appendix 1

This appendix presents a uniform format, known as XHSTT, developed by a group of researchers for characterizing data and solution to the high school timetabling problem. All information provided here to define this format was extracted from (Benchmarking project for (high) school timetabling), where more details can be found.

A.1
XHSTT format

As studies in timetabling problems have advanced, emerged the necessity of a uniform format for managing data, so that researchers could apply their approaches to the same datasets and compare performances and solutions. For this reason a group of researchers started a project, reported in the PATAT conferences of 2008 and 2010, aiming to create a unified format for the school timetabling problem, which would make possible the exchange of benchmarks.

The format was called XHSTT and has a xml-standard. On the website (Benchmarking project for (high) school timetabling) there are available archives and datasets in XHSTT format, an evaluator for instances (by Jeff Kingston) and solutions in XHSTT, and a classification (by Nelishia Pillay) of high school timetabling. Latest update reports there are around 50 datasets available.

Because the format grew out of several years of discussions, it became quite abstract — its tags have very generic names and its structure is very flexible. Consequently, it can be difficult in the beginning to understand XHSTT. Following sections give a short introduction to it. For extensive documentation we refer to (Documentation for XHSTT format).

A.1.1
General structure

The main structure of a XHSTT file is:

```
Archives
    Instances
        Times
        Resources
        Events
        Constraints
    SolutionGroups
        Solutions
            Reports
```

The primary goal of XHSTT is to contain datasets for High School Timetabling. Apart from this, it can contain solutions as well, but the focus will be on the instance itself.

An instance is one occurrence of the high school timetable problem, for a particular school in a particular year (or semester, etc.). It contains 4 groups of items: items related to time, items related to resources, items related to events, and items related to constraints. The first peculiarity of XHSTT is that the first 3 groups (`<Times>`, `<Resources>`, and `<Events>`) contain relatively little information. This means that almost all business logic is carried by the constraints, which define how these previous groups can interact with each other. The syntax of an instance is:

```
Instance Id
    MetaData
    Times
    Resources
    Events
    Constraints
```

In XHSTT notation, the placement on the same line indicates that one category is an attribute of another, indenting indicates that one category is a child of another, + indicates that the immediately following category is optional, and * indicates that the immediately following category may appear zero or more times.

A.1.2 Times

The `Times` section contains four entities: `TimeGroups`, `Weeks`, `Days` and `Times`. A `Day` or `Week` is simply a special `TimeGroup`, which can be added as a property to a `Time`. At all other places it is referred to as `TimeGroup`. The reason for the existence of `Days` and `Weeks` is for displaying only. In the school timetabling problem only one week exists. As it can be guessed, a `TimeGroup` is a set of `Times`. `Times` are assumed to be consecutive as given in the instance.

A.1.3 Resources

The Resources section contains three entities: ResourceTypes, ResourceGroups and Resources. A Resource has exactly one ResourceType. The ResourceType is introduced for displaying (for example: distinguishes teachers from students) and for consistency in assigning resources to events (see the Role section below). ResourceGroups are groups of Resources of the same ResourceType. Common ResourceTypes are Teacher, Room, Class and Student.

A.1.4 Events

The Events section contains three entities: EventGroups, Courses and Events. Courses are special EventGroups. An Event has a property Course. At all other places Courses are referred to as EventGroup. Obviously, an EventGroup is simply a set of Events.

The syntax of Event is

```
Event Id +Color
    Name
    Duration
    +Workload
    +Course
    +Time
    +Resources
    +ResourceGroups
    +EventGroups
```

An event can have two interpretations in XHSTT. The first one is that an Event is a lesson of a fixed Duration. Hence our timetabling problem is to set a begin Time to events, which implies that the Event is planned to this Time, and some consecutive Times if the Duration exceeds 1. Since an Event can have Resources attached to it, these Resources are now busy at these Times. We meet the 2 basic requirements in timetabling, which are made explicit as constraint in XHSTT: we need to assign a (begin)Time to each Event (AssignTimeConstraint) and we have to make sure that Resources are not planned double (AvoidClashesConstraint).

The other meaning of Event is slightly more complicated: an Event can represent all lessons with exactly the same properties, i.e. a course section. These properties are for example the Resources class and teacher. In this case the duration is the total `Duration` required for this class-teacher combination. So now a part of the planning problem is to divide the instance `Event` in solution `Events`; the solution `Events` are the lessons in the first interpretation.

To know which interpretation is valid, a dataset with `Events` of `Durations` greater than 1 will contain one or more `SplitEventsConstraints`, explaining how the instance event can be divided into solution events. If these constraints don't give enough details, one can add `DistributeSplitEventsConstraints` to control the number of solution `Events` of a certain `Duration`.

An example of the first interpretation is when `SplitEventsConstraint` has $MinimumAmount = 1$ and $MaximumAmount = 1$, which means that the instance `Event` should lead to exactly 1 solution `Event`. An example of the second interpretation is if $MinimumAmount = 1$ and $MaximumAmount = 999$, so we can split the instance `Event` to as many solution `Events` as we wish, as long as the sum of `Durations` of the solution `Events` is the `Duration` of the instance `Event` (otherwise the solution is marked invalid).

Roles

In the last subsection we described 2 scheduling decisions that are needed in high school timetabling: creating solution `Events` from instance `Events`, and setting start `Times` to the solution `Events`. The third scheduling decision is the assignment of `Resources` to `Events`. To enable this, `Roles` are added to `Events`. `Roles` are properties of `Events`, and reappear in the Constraints `AssignResourceConstraint`, `PreferResourcesConstraint`, and `AvoidSplitAssignmentsConstraint`. The primary aim of `Roles` is to describe what `Resource` has to be assigned to an `Event`. Mostly, this `Resource` is a room or a teacher. The `Role` always contains a `ResourceType`; the `Resource` assigned to the `Role` should be of this `ResourceType`.

A.1.5 Constraints

Constraints are conditions that should be satisfied whenever it is possible. In XHSTT, even those conditions which are fundamental to any timetabling problem have to be explicitly informed, for example prohibition of resources overbooking. Each constraint has a set of points of application which are instance entities (such as resources or events). Given a solution, a non-negative integer cost is associated with each point of application. A non-zero cost for some point of application indicates that the solution violates the constraint at that point.

In XHSTT constraints exist in two modes: constraints that are `required` (hard constraints) and constraints that are not `required` (soft constraints). All constraints can be in either mode. If all `required` constraints are satisfied, we call the schedule feasible. Violating `required` constraints add costs to the infeasibility value of the solution. The non-required constraints add cost to the objective value. During scheduling (generating solutions) the primary goal is minimizing the infeasibility value, and the secondary goal is minimizing the objective value. Apart from required or not, a constraint also has a `weight`: the generated cost increases linearly with the `weight`.

It is allowed to use constraint of the same type several times. So what we call a "constraint" in fact is a "constraint type", that can reappear with different parameters any number of times.

Many types of constraints are defined, and more may be added in the future. The syntax of constraints that currently appear within the `Constraints` child category of the `Instance` category is

```
Constraints
    *AssignResourceConstraint
    *AssignTimeConstraint
    *SplitEventsConstraint
    *DistributeSplitEventsConstraint
    *PreferResourcesConstraint
    *PreferTimesConstraint
    *AvoidSplitAssignmentsConstraint
    *SpreadEventsConstraint
    *LinkEventsConstraint
    *OrderEventsConstraint
    *AvoidClashesConstraint
    *AvoidUnavailableTimesConstraint
    *LimitIdleTimesConstraint
    *ClusterBusyTimesConstraint
    *LimitBusyTimesConstraint
    *LimitWorkloadConstraint
```

Some properties are common to all constraint types. In general, a constraint has syntax

```
AnyConstraint Id
    Name
    Required
```

```
Weight
CostFunction
AppliesTo
...
```

where `AnyConstraint` stands for any of the child categories of `Constraints` listed above, and . . . stands for additional child categories which vary with the constraint type. As usual, the `Id` attribute is used to reference the constraint from elsewhere in the file, while the `Name` child category is used when printing the constraint in human-readable form.

The evaluation of one constraint at one point of application (that is, the determination of a single cost) proceeds in two stages. In the first stage, a non-negative integer *deviation* is calculated. How this is done depends on the constraint type. The second stage is common to all constraint types. It is influenced by the `Required`, `Weight`, and `CostFunction` child categories. All three have no attributes and no child categories, merely a body. The body of `Required` must be either true or false. The body of `Weight` must be an integer in the range 0 to 1000 inclusive. The body of `CostFunction` must be either “Linear”, “Quadratic” or “Step”. The cost is then calculated by $Cost = Weight{\cdot}CostFunction(deviation)$. That is, the cost function is applied to the deviation, producing an integer which is multiplied by the weight to obtain the cost.

Instances should be encoded on the understanding that violations of hard constraints are serious defects, and that solvers aim to find solutions with very few hard constraint violations. Although the existence of such solutions is not guaranteed, realistic instances should have them. On the other hand, violations of soft constraints are normal and expected.

The syntax of the `AppliesTo` category varies according to each constraint type and can be related to `Events` and `Resources`. Definition for each constraint type or further details can be found at (Documentation for constraints in XHSTT format).

A.2
Converting XHSTT problem into Trieda’s problem

Although there is not an exact correspondence between the XHSTT format and the one considered in this dissertation, in the sense that both lack features of the other, an attempt of converting a XHSTT problem into Trieda’s problem was made.

Table A.1 organizes the correspondence that can be done between entities and constraints of both problems.

Table A.1: Correspondence of data between XHSTT and Trieda

TRIEDA	$\longleftrightarrow$	XHSTT
Course Section	$\leftrightarrow$	EventGroup Course
Student (single)	$\leftrightarrow$	Resource of type Student
Student (class meaning)	$\leftrightarrow$	Resource of type Class
Professor	$\leftrightarrow$	Resource of type Teacher
Classroom	$\leftrightarrow$	Resource of type Room
Calender Timetable	$\leftrightarrow$	TimeGroups
Shift	$\leftrightarrow$	TimeGroups
Session of day	$\leftrightarrow$	TimeGroups
Time slot	$\leftrightarrow$	Time
Student-Demand	$\leftrightarrow$	Event + AssignResourceConstraint
Credits Split Rules	$\leftrightarrow$	SplitEventsConstraint, DistributeSplitEventsConstraint and SpreadEventsConstraint
Capability for teaching courses	$\leftrightarrow$	Event with pre-assigned teacher, PreferResourcesConstraint
Professor preference for teaching courses	$\leftrightarrow$	PreferResourcesConstraint
Possible classrooms for each course	$\leftrightarrow$	Event with pre-assigned room, PreferResourcesConstraint
Time Availability	$\leftrightarrow$	Event with pre-assigned time, PreferTimesConstraint and AvoidUnavailableTimesConstraint
Compact students timetables	$\leftrightarrow$	LimitIdleTimesConstraints
Compact professors timetables	$\leftrightarrow$	LimitIdleTimesConstraints
Minimum and maximum professor workload	$\leftrightarrow$	LimitWorkloadConstraint
No time slots overlapping	$\leftrightarrow$	AvoidClashesConstraint
Minimum nr of credits in professor's day	$\leftrightarrow$	LimitBusyTimesConstraint
Maximum nr of busy days per professor	$\leftrightarrow$	ClusterBusyTimesConstraint
One professor for each course section	$\leftrightarrow$	AvoidSplitAssignmentsConstraint
Same classroom for each course section	$\leftrightarrow$	AvoidSplitAssignmentsConstraint

Suppose the following Event:

```
Event Id
    Name
    Duration
    Course
    ResourceGroups
      Class
      Teacher
      Room
```

This `Event` gathers together a `Course` with some `Duration` and the resources types `Class`, `Teacher` and `Room`. When linked to the `AssignResourceConstraint`, this `Event` means that the specified `Class` demands lessons with total `Duration` of the specified `Course` with some resource of type `Teacher` and with some resource of type `Room`. In this dissertation, this is interpreted as a *Student-Demand*.

There are three ways for setting *available times* for resources. The most basic way is to use `AvoidUnavailableTimesConstraint` to inform which times are not available for each resource. Another way is to pre-assign a time to an event, which means that there is actually no time assignment decision to be made for this event. Lastly, `PreferTimesConstraint` can be used when some time slots are preferred to others, although all them are available.

Very similarly, there are two ways for setting *courses capabilities* for professors and *possible classrooms* for each course: pre-assigning respectively teachers and rooms to events; and using `PreferResourcesConstraint` to inform that some resources are preferred to others.

A.3 Suggestions for expansion of XHSTT format

The format XHSTT is still not complete — it has been developed and improved in the last few years, and there are several requirements that are not covered yet. Following, some restrictions are suggested to be added to XHSTT. They are used in this dissertation and have proven themselves essential so that actual deployable solutions can be found.

A.3.1 Travel time between different blocks

A known requirement that is not considered by the current format is to handle different and connected blocks (called “campuses” in XHSTT’s documentation).

Maybe students and/or teachers have to travel between different blocks. In these situations, the time necessary for moving between blocks must be considered, so that a actual solution can be found.

The suggested syntax is:

```
DisplacementTimeConstraint Id
    Name
    Required
    Weight
    CostFunction
    AppliesTo
    TimeGroups
```

Its `AppliesTo` category has syntax

```
AppliesTo
    ResourceTuples
```

where `ResourceTuples` has syntax

```
ResourceTuples
    *ResourceTuple
```

and `ResourceTuple` has syntax

```
ResourceTuple
    FirstResourceGroup Reference
    SecondResourceGroup Reference
    MoveResourceGroup Reference
    MinSeparation
```

`FirstResourceGroup` and `SecondResourceGroup` contain references to two resource groups that should be apart by a minimum value when assigned to a resource of `MoveResourceGroup`. The idea is that each block has a resource group gathering its rooms; then `FirstResourceGroup` and `SecondResourceGroup` identify a pair of blocks such that the minimum displacement time between them is `MinSeparation`. Each `MoveResourceGroup` refers to a group of movable resources, that is, teachers, students or classes.

The syntax of the `TimeGroups` child category is

```
TimeGroups
    *TimeGroup
```

where TimeGroup has syntax

```
TimeGroup Reference
```

and references a time group, which may be a Day.

For each resource of MoveResourceGroup, assignments made to resources of FirstResourceGroup and SecondResourceGroup in the same TimeGroup should respect a minimum separation MinSeparation (in number of time slots), so that the movable resource is capable to travel between resources.

Each resource of MoveResourceGroup is said to be one point of application. The deviation at one point is the total number of times necessary for displacement that was violated.

A.3.2 Limited number of blocks in teachers/classes timetables

Another important restriction discussed in this dissertation is to limit the number of blocks assigned to a movable resource along its day or week.

The suggested syntax is:

```
MaxNrGroupResourcesThroughTime Id
    Name
    Required
    Weight
    CostFunction
    AppliesTo
    TimeGroups
```

Its AppliesTo category has syntax

```
AppliesTo
    ResourceTuples
```

where the syntax of ResourceTuples is

```
ResourceTuples
    *ResourceTuple
```

and the syntax of ResourceTuple is

```
ResourceTuple
    *ResourceGroup Reference
    MoveResourceGroup Reference
    NrMaxResourceGroup
```

*ResourceGroup is a list that contains references to resource groups that should be limited by a maximum value when assigned to a resource of MoveResourceGroup. The idea is that each block has a resource group gathering its rooms. Each MoveResourceGroup refers to a group of movable resources, that is, teachers, students or classes. For each movable resource, the number of different blocks that are assigned to it during some period of time should not exceed NrMaxResourceGroup.

The syntax of the TimeGroups child category is

```
TimeGroups
    *TimeGroup
```

where TimeGroup has syntax

```
TimeGroup Reference
```

and references a time group, which may be a Day or Week. Each TimeGroup is a period of time to be considered by the restriction. When considering a Day, it is possible to require that a movable resource is not assigned to more than NrMaxResourceGroup blocks in that Day. Analogous restrictions can be set if one varies the TimeGroup, such as considering Week, MondayMorning, etc.

Each resource of MoveResourceGroup is one point of application and the deviation is the sum of extra number of blocks that was used in each constraint.

B
Appendix 2

B.1
Solution quality of individual computational experiments

Following are registered the individual results for all computational experiments performed and used as base for approaches' analysis at Chapter 5. As previously explained, each scenario was solved with four different approaches combinations and, due to the random element of the methods, especially of the polishing, each pair [scenario+approaches combination] was performed 5 times.

B.1.1
Scenario A

Goal Programming + Polish

Table B.1: Solution quality of tests for scenario A solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	318	367	364	345	358
Prof Gaps ign. displac	161	192	189	180	198
Prof long displac	46	56	59	53	53
Used professors	368	380	377	376	383
Real professors	277	277	277	277	277
Virtual professors	91	103	100	99	106
Credits assigned to prof	5582	5585	5594	5585	5578
↪ to real prof	5430	5390	5419	5409	5393
↪ to virtual prof	152	195	175	176	185
Courses sections	5465	5467	5476	5468	5461
↪ with real prof	5314	5274	5303	5293	5277
↪ with virtual prof	151	193	173	175	184
% of satisfied credits	98.91	98.98	99.14	98.97	98.86
Satisfied stud-dem	5551	5554	5563	5554	5548
Non-satisfied stud-dem	62	59	50	59	65
Total Run Time	9h10'57"	9h44'02"	9h03'34"	9h51'48"	9h15'11"

Goal Programming + No Polish

Table B.2: Solution quality of tests for scenario A solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	1	0	0	0	0
Prof Gaps ign. displac	1	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	95	95	92	96	96
Real professors	89	87	86	89	89
Virtual professors	6	8	6	7	7
Credits assigned to prof	446	443	467	456	456
$\hookrightarrow$ to real prof	430	413	443	436	436
$\hookrightarrow$ to virtual prof	16	30	24	20	20
Courses sections	438	435	461	450	450
$\hookrightarrow$ with real prof	422	405	437	430	430
$\hookrightarrow$ with virtual prof	16	30	24	20	20
% of satisfied credits	7.69	7.64	8.05	7.86	7.86
Satisfied stud-dem	438	435	461	450	450
Non-satisfied stud-dem	5175	5178	5152	5163	5163
Total Run Time	10h14’18”	10h14’19”	10h12’35”	10h09’47”	10h10’10”

Multi-objective function + Polish

Table B.3: Solution quality of tests for scenario A solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	397	364	345	400	415
Prof Gaps ign. displac	223	208	197	252	273
Prof long displac	43	54	45	35	39
Used professors	379	381	364	374	376
Real professors	277	277	277	277	277
Virtual professors	102	104	87	97	99
Credits assigned to prof	5598	5582	5591	5608	5596
$\hookrightarrow$ to real prof	5401	5395	5420	5409	5399
$\hookrightarrow$ to virtual prof	197	187	171	199	197
Courses sections	5471	5460	5467	5481	5471
$\hookrightarrow$ with real prof	5275	5275	5297	5283	5276
$\hookrightarrow$ with virtual prof	196	185	170	198	195
% of satisfied credits	98.67	98.64	98.66	98.86	98.79
Satisfied stud-dem	5537	5534	5536	5548	5543
Non-satisfied stud-dem	76	79	77	65	70
Total Run Time	2h28’58”	4h39’37”	4h29’10”	4h34’53”	3h50’19”

Multi-objective function + No Polish

Table B.4: Solution quality of tests for scenario A solved with multi-objective function, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	249	234	244	242	227
Prof Gaps ign. displac	245	229	235	228	221
Prof long displac	0	1	1	2	1
Used professors	96	97	97	96	95
Real professors	88	90	90	89	88
Virtual professors	8	7	7	7	7
Credits assigned to prof	275	271	269	271	274
↪ to real prof	256	258	254	252	253
↪ to virtual prof	19	13	15	19	21
Courses sections	269	265	263	265	268
↪ with real prof	250	252	248	246	247
↪ with virtual prof	19	13	15	19	21
% of satisfied credits	4.74	4.67	4.64	4.67	4.72
Satisfied stud-dem	269	265	263	265	268
Non-satisfied stud-dem	5344	5348	5350	5348	5345
Total Run Time	8h22’47”	8h19’33”	8h16’13”	8h15’59”	8h15’17”

B.1.2 Scenario B1

Goal Programming + Polish

Table B.5: Solution quality of tests for scenario B1 solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	434	497	547	555	512
Prof Gaps ign. displac	290	327	416	406	362
Prof long displac	128	135	124	118	133
Used professors	644	685	667	667	656
Real professors	471	471	471	471	471
Virtual professors	173	214	196	196	185
Credits assigned to prof	9269	9269	9268	9269	9268
↪ to real prof	8897	8816	8871	8862	8865
↪ to virtual prof	372	453	397	407	403
Courses sections	9028	9028	9027	9028	9027
↪ with real prof	8660	8579	8634	8626	8629
↪ with virtual prof	368	449	393	402	398
% of satisfied credits	99.98	99.98	99.97	99.98	99.98
Satisfied stud-dem	9182	9182	9181	9182	9182
Non-satisfied stud-dem	1	1	2	1	1
Total Run Time	10h08’07”	9h50’38”	10h22’00”	10h14’27”	10h09’44”

Goal Programming + No Polish

Table B.6: Solution quality of tests for scenario B1 solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
↪ to real prof	0	0	0	0	0
↪ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
↪ with real prof	0	0	0	0	0
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	9183	9183	9183	9183	9183
Total Run Time	9h09'03"	9h01'10"	9h12'00"	9h08'50"	9h05'37"

Multi-objective function + Polish

Table B.7: Solution quality of tests for scenario B1 solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	520	493	435	440	492
Prof Gaps ign. displac	386	368	317	299	339
Prof long displac	115	92	101	108	107
Used professors	662	686	685	687	681
Real professors	471	471	471	471	471
Virtual professors	191	215	214	216	210
Credits assigned to prof	9306	9307	9311	9317	9311
↪ to real prof	8903	8838	8846	8830	8845
↪ to virtual prof	403	469	465	487	466
Courses sections	9058	9060	9061	9070	9063
↪ with real prof	8659	8595	8600	8587	8601
↪ with virtual prof	399	465	461	483	462
% of satisfied credits	99.98	99.98	99.97	99.98	99.98
Satisfied stud-dem	9182	9182	9181	9182	9182
Non-satisfied stud-dem	1	1	2	1	1
Total Run Time	4h57'04"	5h06'12"	5h07'31"	5h08'09"	5h03'00"

Multi-objective function + No Polish

Table B.8: Solution quality of tests for scenario B1 solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
$\hookrightarrow$ to real prof	0	0	0	0	0
$\hookrightarrow$ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
$\hookrightarrow$ with real prof	0	0	0	0	0
$\hookrightarrow$ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	9183	9183	9183	9183	9183
Total Run Time	8h36’30”	8h37’10”	8h41’06”	8h30’12”	8h34’13”

B.1.3 Scenario B2

Goal Programming + Polish

Table B.9: Solution quality of tests for scenario B2 solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	391	493	383	455	480
Prof Gaps ign. displac	301	392	279	355	380
Prof long displac	80	87	89	75	85
Used professors	636	639	637	644	637
Real professors	471	471	471	471	471
Virtual professors	165	168	166	173	166
Credits assigned to prof	9159	9149	9157	9160	9153
↪ to real prof	8834	8845	8858	8841	8839
↪ to virtual prof	325	304	299	319	314
Courses sections	8918	8908	8916	8919	8912
↪ with real prof	8599	8610	8621	8604	8604
↪ with virtual prof	319	298	295	315	308
% of satisfied credits	98.83	98.72	98.81	98.83	98.77
Satisfied stud-dem	9073	9063	9071	9073	9067
Non-satisfied stud-dem	110	120	112	110	116
Total Run Time	8h44'26"	9h22'32"	9h25'12"	10h19'32"	9h36'43"

Goal Programming + No Polish

Table B.10: Solution quality of tests for scenario B2 solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
↪ to real prof	0	0	0	0	0
↪ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
↪ with real prof	0	0	0	0	0
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	9183	9183	9183	9183	9183
Total Run Time	8h41'38"	8h42'27"	8h30'28"	8h50'44"	8h51'01"

Multi-objective function + Polish

Table B.11: Solution quality of tests for scenario B2 solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	479	511	521	448	555
Prof Gaps ign. displac	388	380	400	343	433
Prof long displac	76	82	82	75	74
Used professors	655	645	649	659	657
Real professors	470	470	469	470	470
Virtual professors	185	175	180	189	187
Credits assigned to prof	9205	9196	9180	9213	9203
↪ to real prof	8838	8816	8821	8850	8818
↪ to virtual prof	367	380	359	363	385
Courses sections	8955	8947	8930	8963	8955
↪ with real prof	8592	8571	8577	8606	8575
↪ with virtual prof	363	376	353	357	380
% of satisfied credits	98.89	98.75	98.66	98.93	98.79
Satisfied stud-dem	9079	9065	9057	9082	9069
Non-satisfied stud-dem	104	118	126	101	114
Total Run Time	4h45'30"	5h00'54"	4h50'33"	4h47'59"	4h59'23"

Multi-objective function + No Polish

Table B.12: Solution quality of tests for scenario B2 solved with multi-objective function, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
↪ to real prof	0	0	0	0	0
↪ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
↪ with real prof	0	0	0	0	0
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	9183	9183	9183	9183	9183
Total Run Time	8h25'15"	8h21'04"	8h20'44"	8h22'09"	8h24'56"

B.1.4 Scenario C

Goal Programming + Polish

Table B.13: Solution quality of tests for scenario C solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	89	120	80	102	95
Prof Gaps ign. displac	76	108	74	92	84
Prof long displac	17	16	12	14	14
Used professors	260	260	259	263	259
Real professors	243	243	243	243	243
Virtual professors	17	17	16	20	16
Credits assigned to prof	2646	2646	2646	2646	2646
↪ to real prof	2621	2621	2622	2617	2620
↪ to virtual prof	25	25	24	29	26
Courses sections	2584	2584	2584	2584	2584
↪ with real prof	2559	2559	2560	2555	2558
↪ with virtual prof	25	25	24	29	26
% of satisfied credits	99.93	99.93	99.93	99.93	99.93
Satisfied stud-dem	2624	2624	2624	2624	2624
Non-satisfied stud-dem	1	1	1	1	1
Total Run Time	5h45’09”	6h08’04”	5h40’37”	5h35’34”	5h48’13”

Goal Programming + No Polish

Table B.14: Solution quality of tests for scenario C solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
↪ to real prof	0	0	0	0	0
↪ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
↪ with real prof	0	0	0	0	0
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	2625	2625	2625	2625	2625
Total Run Time	8h09'59"	8h05'39"	8h11'00"	8h05'50"	8h07'27"

Multi-objective function + Polish

Table B.15: Solution quality of tests for scenario C solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	192	240	204	186	207
Prof Gaps ign. displac	176	214	193	181	198
Prof long displac	12	29	8	10	7
Used professors	256	260	261	259	258
Real professors	243	243	243	243	243
Virtual professors	13	17	18	16	15
Credits assigned to prof	2657	2658	2657	2653	2660
↪ to real prof	2632	2630	2625	2628	2638
↪ to virtual prof	25	28	32	25	22
Courses sections	2591	2591	2593	2589	2594
↪ with real prof	2566	2563	2562	2564	2572
↪ with virtual prof	25	28	31	25	22
% of satisfied credits	99.93	99.93	99.93	99.93	99.93
Satisfied stud-dem	2624	2624	2624	2624	2624
Non-satisfied stud-dem	1	1	1	1	1
Total Run Time	2h28'17"	2h04'03"	1h49'47"	2h43'24"	2h50'15"

Multi-objective function + No Polish

Table B.16: Solution quality of tests for scenario C solved with multi-objective function, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	0	0	0	0	0
Real professors	0	0	0	0	0
Virtual professors	0	0	0	0	0
Credits assigned to prof	0	0	0	0	0
$\hookrightarrow$ to real prof	0	0	0	0	0
$\hookrightarrow$ to virtual prof	0	0	0	0	0
Courses sections	0	0	0	0	0
$\hookrightarrow$ with real prof	0	0	0	0	0
$\hookrightarrow$ with virtual prof	0	0	0	0	0
% of satisfied credits	0	0	0	0	0
Satisfied stud-dem	0	0	0	0	0
Non-satisfied stud-dem	2625	2625	2625	2625	2625
Total Run Time	8h06’20”	8h11’10”	8h10’14”	8h06’58”	8h07’07”

B.1.5 Scenario D

Goal Programming + Polish

Table B.17: Solution quality of tests for scenario D solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	23	22	24	21	22
Prof Gaps ign. displac	18	16	17	15	17
Prof long displac	0	0	0	0	0
Used professors	68	71	70	68	70
Real professors	58	58	58	58	58
Virtual professors	10	13	12	10	12
Credits assigned to prof	648	649	649	647	648
↪ to real prof	631	630	630	629	629
↪ to virtual prof	17	19	19	18	19
Courses sections	648	649	649	647	648
↪ with real prof	631	630	630	629	629
↪ with virtual prof	17	19	19	18	19
% of satisfied credits	99.54	99.54	99.69	99.39	99.54
Satisfied stud-dem	650	650	651	649	650
Non-satisfied stud-dem	3	3	2	4	3
Total Run Time	3h29'52"	5h30'14"	4h38'04"	5h06'52"	4h46'44"

Goal Programming + No Polish

Table B.18: Solution quality of tests for scenario D solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	28	51	56	55	24
Prof Gaps ign. displac	22	47	52	51	20
Prof long displac	0	0	0	0	0
Used professors	69	68	67	69	68
Real professors	58	58	58	58	58
Virtual professors	11	10	9	11	10
Credits assigned to prof	649	649	648	649	649
↪ to real prof	631	630	631	631	630
↪ to virtual prof	18	19	17	18	19
Courses sections	649	649	648	649	649
↪ with real prof	631	630	631	631	630
↪ with virtual prof	18	19	17	18	19
% of satisfied credits	99.69	99.69	99.54	99.69	99.69
Satisfied stud-dem	651	651	650	651	651
Non-satisfied stud-dem	2	2	3	2	2
Total Run Time	4h55'27"	4h16'57"	4h32'30"	4h47'00"	4h40'17"

Multi-objective function + Polish

Table B.19: Solution quality of tests for scenario D solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	22	29	25	26	33
Prof Gaps ign. displac	20	26	22	22	28
Prof long displac	0	0	0	0	0
Used professors	68	67	67	67	67
Real professors	58	58	58	58	58
Virtual professors	10	9	9	9	9
Credits assigned to prof	648	648	649	649	294
$\hookrightarrow$ to real prof	631	631	631	631	276
$\hookrightarrow$ to virtual prof	17	17	18	18	18
Courses sections	648	648	649	649	649
$\hookrightarrow$ with real prof	631	631	631	631	631
$\hookrightarrow$ with virtual prof	17	17	18	18	18
% of satisfied credits	99.54	99.54	99.69	99.69	99.69
Satisfied stud-dem	650	650	651	651	651
Non-satisfied stud-dem	3	3	2	2	2
Total Run Time	1h01'00"	0h39'54"	1h14'40"	1h27'03"	1h17'14"

Multi-objective function + No Polish

Table B.20: Solution quality of tests for scenario D solved with multi-objective function, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	84	24	24	22	28
Prof Gaps ign. displac	69	23	20	18	22
Prof long displac	0	0	0	0	0
Used professors	74	66	66	68	70
Real professors	58	58	57	58	58
Virtual professors	16	8	9	10	12
Credits assigned to prof	651	649	649	649	649
$\hookrightarrow$ to real prof	608	631	631	630	630
$\hookrightarrow$ to virtual prof	43	18	18	19	19
Courses sections	651	649	649	649	649
$\hookrightarrow$ with real prof	608	631	631	630	630
$\hookrightarrow$ with virtual prof	43	18	18	19	19
% of satisfied credits	100	99.69	99.69	99.69	99.69
Satisfied stud-dem	653	651	651	651	651
Non-satisfied stud-dem	0	2	2	2	2
Total Run Time	2h33'21"	3h01'46"	1h53'40"	1h25'53"	2h17'24"

B.1.6 Scenario E

Goal Programming + Polish

Table B.21: Solution quality of tests for scenario E solved with goal programming, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	35	35	35	35	35
Real professors	35	35	35	35	35
Virtual professors	0	0	0	0	0
Credits assigned to prof	434	434	434	434	434
↪ to real prof	434	434	434	434	434
↪ to virtual prof	0	0	0	0	0
Courses sections	434	434	434	434	434
↪ with real prof	434	434	434	434	434
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	100	100	100	100	100
Satisfied stud-dem	443	443	443	443	443
Non-satisfied stud-dem	0	0	0	0	0
Total Run Time	0h29’11”	0h36’58”	0h37’33”	0h34’50”	0h35’32”

Goal Programming + No Polish

Table B.22: Solution quality of tests for scenario E solved with goal programming, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	0	0	0	0
Prof Gaps ign. displac	0	0	0	0	0
Prof long displac	0	0	0	0	0
Used professors	35	35	35	35	35
Real professors	35	35	35	35	35
Virtual professors	0	0	0	0	0
Credits assigned to prof	434	434	434	434	434
↪ to real prof	434	434	434	434	434
↪ to virtual prof	0	0	0	0	0
Courses sections	434	434	434	434	434
↪ with real prof	434	434	434	434	434
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	100	100	100	100	100
Satisfied stud-dem	443	443	443	443	443
Non-satisfied stud-dem	0	0	0	0	0
Total Run Time	0h39’02”	0h33’00”	0h40’26”	0h35’54”	0h50’57”

Multi-objective function + Polish

Table B.23: Solution quality of tests for scenario E solved with multi-objective function, polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	0	1	0	0	1
Prof Gaps ign. displac	0	1	0	0	1
Prof long displac	0	0	0	0	0
Used professors	35	35	35	35	35
Real professors	35	35	35	35	35
Virtual professors	0	0	0	0	0
Credits assigned to prof	435	434	435	434	435
↪ to real prof	435	434	435	434	435
↪ to virtual prof	0	0	0	0	0
Courses sections	435	434	435	434	435
↪ with real prof	435	434	435	434	435
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	100	100	100	100	100
Satisfied stud-dem	443	443	443	443	443
Non-satisfied stud-dem	0	0	0	0	0
Total Run Time	0h42’52”	1h32’14”	0h57’16”	1h52’03”	0h42’27”

Multi-objective function + No Polish

Table B.24: Solution quality of tests for scenario E solved with multi-objective function, no polishing method and without professor priority

Features	Rounds				
	1	2	3	4	5
Prof Gaps	47	0	1	0	1
Prof Gaps ign. displac	43	0	1	0	1
Prof long displac	0	0	0	0	0
Used professors	35	35	35	35	35
Real professors	35	35	35	35	35
Virtual professors	0	0	0	0	0
Credits assigned to prof	437	435	435	434	434
↪ to real prof	437	435	435	434	434
↪ to virtual prof	0	0	0	0	0
Courses sections	437	435	435	434	434
↪ with real prof	437	435	435	434	434
↪ with virtual prof	0	0	0	0	0
% of satisfied credits	100	100	100	100	100
Satisfied stud-dem	443	443	443	443	443
Non-satisfied stud-dem	0	0	0	0	0
Total Run Time	2h08'47"	1h45'31"	1h16'37"	1h37'00"	1h39'27"