UCL
Université
catholique
de Louvain

LOUVAIN
School of Management

Louvain School of Management

Application of integer programming techniques to facilitate the production of examination timetables at FIAL faculty

Research Master's Thesis submitted by
Virgile Pypaert

With the view of getting the degree in
Master 120 crédits en ingénieur de gestion, à finalité spécialisée

Supervisor
Mathieu Van Vyve

Academic Year 2017–2018

Application of integer programming techniques to facilitate the production of examination timetables at FIAL faculty

Virgile Pypaert
Louvain School of Management
Master thesis under supervision of Prof. Mathieu Van Vyve

Academic Year 2017–2018

Acknowledgements

I dedicate this section to all the people who contributed in one way or another to the completion of the work described in this thesis.

Firstly, I would like to thank my thesis supervisor Prof. Mathieu Van Vyve whose door was constantly open whenever I was in need for an insight or had a question about my research or writing. He steered me in the right direction when I needed it and opened my eyes on several aspects that deserved to be covered in this paper.

Secondly, I could not miss to acknowledge the contribution of FIAL faculty employees who helped me at several occasions and more specifically Mrs. Julie Thirionet who took the time to explain the operations and described the current problems and possibilities to be considered in my thesis. Her deep knowledge of the process was beneficial to the present work.

Finally, I must express my gratitude to my parents and to my friends who provided me a constant support throughout my years of study and through the process of building this thesis. I am sure that this accomplishment would not have been possible without them. I would like to thank as well the one and all who have contributed directly or indirectly to this venture.

Thank you.

Virgile Pypaert

Executive Summary

Besides being an omen of hard times to come for the students, examination timetables are the result of a long and tedious work by the faculty's staff who is responsible for their generation. From the enrolments until the delivery of the first draft to the students, up to one whole month of work can be needed by the employees of the department. This last statement is far from being a criticism of their aptitude given the complexity to solve such problems even when using considerable computational power.

In fact, the essence of the problem lies in the attribution of exclusive resources contained in multiple sets (students, professors, time slots, rooms) to events that are meant to take place (the exams) while taking care of various constraints and requirements that have to be fulfilled to implement the solution in real terms. These constraints vary from institution to institution even if all agree on the impossibility to take two exams simultaneously.

When this thesis was written, most of the operations regarding the creation of examination timetables were performed manually at FIAL faculty. Our goal is to determine if the use of integer programming to transpose the problem is doable, facilitates the generation of solutions and improves their quality in comparison with the manual approach. Based on the observations of the experiment, some future points of research will be suggested.

Contents

List of Figures

List of Tables

Introduction

This introductory chapter serves as a bird's eye view of the whole thesis, approaching the different topics that will be covered in the whole dissertation to attempt to build a solution that improves the current situation regarding examination timetabling. We will specify here the goals of this thesis precising the scope, the research questions and the global structure of the following chapters.

Scope and goals of the study

Our title, "Application of integer programming techniques to facilitate the production of examination timetables at FIAL faculty", is quite concise about the main goal of this study: analyse whether programming techniques can facilitate the production of examination timetables at the FIAL faculty and to what extent does it affect the quality of the sessions that to be taken by the students.

As we do not have the total suppression of manual handling as an objective, we first try to review the efficiency of using a computer based-model to create feasible timetables. In such conditions, our goal is to test whether our model is capable of handling simultaneously the most frequent requirements that a faculty scheduler could face while finding a solution that is at least qualitatively acceptable as well.

The implementation of our model needs to be considered as well in the scope of our problem. Indeed, while the end-user should be fluent with the different concepts and requirements of an examination timetable, he should not have to follow long and tedious training to use the tools that are put at his disposal. Following that, methodologies that cannot be reproduced on multiple datasets without heavy tweaking and transformation should be excluded from the candidates.

Research questions

To achieve these objectives, we will analyse in this thesis the aspects relative to the three research questions presented below:

- What is the most appropriate computer-based technique that could be used to build examination timetables at FIAL faculty?
- What sort of limitations do we face while looking at the problem complexity? What is the impact on the solution found?

- How easy would it be to alter the current procedure to make use of our model? What kind of improvements can be expected from using such methods? Who will benefit the most from it?

Structure of the dissertation

This section provides information about the structure that will be throughout the rest of this dissertation and define how each part and chapter contribute to the global reasoning of our choice to use of integer programming techniques to solve that kind of problems. This thesis is broken down in two parts and five different chapters.

- The first part is a theoretical review of the concepts and problems specific to the generation of examination timetables.
  - The first chapter defines shortly the key aspects relative to (examination) timetabling problems, it includes: definitions of the most important terms used, examples of other type of timetabling problems, their mathematical complexity, handling methods through time, the most common model requirements, the most obvious stakeholders and how the quality of a solution can be quantified.
  - The second chapter will be about exploring the most popular techniques that are the most frequently encountered to cope with that type of problems (both exact and heuristics methods). This exploration will help as well to justify the use of integer programming for our application to the FIAL faculty in the next part.
- The second part is about the application of the previously-explored theory in the context of the FIAL faculty.
  - The third chapter will cover the building the model by defining the used simplifications, the different requirement both statutory and tacit and finally an integer programming formulation that considers all the specifications and that will be tested afterwards using FIAL datasets will be delivered.
  - The fourth chapter provides information about the way AIMMS has been to perform the tests and how the data must be treated to create the various data inputs required in our model.
  - The final chapter will discuss the different tests we did on the five previous datasets starting from a simple resolution of the problem to the analysis of the variation of several parameters that could vary (the existence of minor slots and the extension or reduction of the number of slots).

The whole will be followed by a conclusion summarizing the findings of our research and experimentations. We deliver other points that could be studied in an extension of this work as well.

I

Theoretical review

1. Examination timetabling

Carter, Laporte and Lee (1996) define **examination timetabling** as a problem where the objective is to assign a set of exams into a limited number of timeslots (and rooms) while taking into account certain requirements and limitations expressed using constraints. In its most basic form, it should not allow (or at least avoid a much as possible) cases where students would have to pass several exams at the same time, on top of that, several optional side-constraints can be taken care of as well (Burke, Jackson, Kingston & Weare, 1997, p. 1).

1.1. Timetabling vs. Scheduling

Even if timetabling and scheduling sound like comparable concepts to most of the people, in the scientific literature, several distinctions are existing in their definitions and require to be ironed out before starting any further investigation.

The range of definitions of **timetabling** is quite wide among the scientific literature. The most basic one defines it as a representation of "when particular events are to take place" without considering other aspects such as the allocation of resources (Wren, 1996, p. 46). On the other side, Burke, Bykov, Newall and Petrovic (2004) define it as a problem of allocating a set of events/meeting requiring some resources among a set of slots under some constraints that should be fulfilled "as far as possible". What is defined in the previous sentence as a 'resource' is relatively vague and could include: people (an individual or a group), rooms (e.g. a specific lab), machinery and equipment (e.g. a projector, an assembly line...), etc. (Burke et al., 1997, p. 1). Constraints are described by Wren (1996, p. 48) as a frame of rules defining the possible (physically or legally) relationships between objects inside a certain pattern. They are simultaneously a part of the problem and a direction for the solver to reach a solution.

In contrast, **scheduling** refers most exclusively to the assignment of entities (events, tasks, people) within a "pattern in space-time" (Landa Silva, Burke & Petrovic, 2004). This addition of a systematic spatial criterion on top of the other 'common ones' (event, time and constraints) fully reflects the type of problem that will be studied in this thesis. Indeed, a proper model should be able to take into account the availability of spatial resources in order to build an examination timetable that could be used in reality (Wren, 1996, p. 48).

After comparing the two directions, the activities of drawing university examination timetables (and by extension course timetables) is closer to a scheduling activity than the *simple* timetabling one (Wren, 1996, p. 49). Further down in this paper, when

timetabling problem is being addressed, we will express it with Burke's definition in mind.

1.1.1. Examples of timetabling problems

Timetabling problems are encountered frequently in our daily lives, whatever the kind of activity we might be doing. Even if we focus here on an examination timetabling problem, the literature tends to split them inside four major categories: personnel rostering, transportation scheduling, sports timetabling and educational timetabling (Qu, Burke, McCollum, Merlot & Lee, 2009, p. 55).

Personnel rostering Every organisation has to build regularly rosters to determine when each member of the staff is required to be on duty in order to perform the activities. The problem seems easy in cases where people work on the clock such as in offices but is more complex when the work is organised in shifts and when the workers' qualification is unequal and thus that specific tasks require specific operators. The most common cases found in the scientific papers include nurse in hospitals and shifts in factories.

Transportation scheduling By nature, transportation services require to assign and use resources at their disposal in the most efficient way possible (or at least try to limit the inefficiencies). This affirmation applies to both public transportation (bus, train, boat, air...) and to related problems (e.g. freight) as well. According to the mode of transportation used, resources to be taken care of could include: the mutual fitting of timetables in multimodal hubs, the rolling stock to be used, the time window when a vehicle could circulate on a track or an airway, the driver who will perform the job, the variation of frequency during peak and off-peak hours, etc.

Sports timetabling As surprising as it may seem at first sight, the logistics required behind some team sport competitions is huge regarding its combinatorial possibilities whatever the level at which it is played. For instance, in a competition composed of 16 teams (such as in Belgian First Division of football), the number of matches to be scheduled reaches 240 with return matches (16×15, one not be playing against itself and one meeting the other in each own stadium) and 120 otherwise. Given the number of rules to be fulfilled at the same time (availability, number of events per period, lapse between events...), the complexity of the problem gets quite significant.

Educational timetabling, a genre of its own ? Let's now focus on timetabling practices that are encountered in educational entities ranging from elementary schools up to the higher-education system. Within the educational timetabling class, it is possible to distinguish 5 major categories of timetabling sub-problems that need to be solved by such institutions (schools or universities): Faculty timetabling, Class-teacher timetabling, Course scheduling, Classroom assignment and Examination timetabling (Bardadym, 1996, p. 24).

- **Faculty timetabling** designates the assignation of the different courses to the teachers, being subject to some constraints such as their area of competence (an English teacher is not fit to teach Maths for instance).
- **Class-teacher timetabling**, in situations where entire classes follow the same courses, it is used to attribute each teacher (who teaches a specific course) to the different classes while making sure that each side has no more than one event at once and that the number of lectures is respected.
- **Course scheduling**, in cases of non-fixed curricula, manages to build a timetable where each student can follow each lecture programmed for the courses he selected $\rightarrow$ generation of a conflict-free schedule.
- **Classroom assignment** refers to the attribution of rooms to lectures that have been scheduled using the previously mentioned "class-teacher timetabling" and "course scheduling" $\rightarrow$ a room cannot hold more than one event per period.
- **Examination timetabling** is used to assign each examination from a set while making sure that no student (or class) have more than one exam simultaneously.

Even if course scheduling and examination timetabling share similar requirements (no conflict), it does not seem that they can be interchanged for all that, given that the objective and constraints might be different (Bergmann, Fischer & Zurheide, 2014, p. 83; Schaerf, 1999, p. 108). For instance, a desirable outcome of the first is a tight schedule with as few gaps as possible but on the other side enough time should be provided to study for the examination; a lecture must happen in a single location while an exam can be split in different rooms; a course means multiple lectures but (usually) a single exam; etc. (Burke et al., 1997, p. 1).

One attempt was documented by Dimopoulou and Miliotis (2001) who tried to build examination timetables by duplicating a weekly course schedule for the length of the session. Starting from a situation where duplicates would exist for each examination, they are sequentially removed until a single slot remains attributed to each exam while making sure that no simultaneous event occurs inside a programme. It involved several issues though: (1) the length of the session is dependent on the number of classes on a day; (2) course schedules need to be conflict-free (which is the case for most of the

students, but not an absolute requirement though); (3) less students can fit in a room during an examination than during a class.

1.2. Complexity of timetabling problems

The complexity designates the difficulty that is required to solve the instances of a particular problem. Decisions problems (Yes or No) are often ranked according to their complexity using the $\mathcal{P}$ and $\mathcal{NP}$ classes (Weisstein, 2002):

- $\mathcal{P}$-Problems can be solved using an algorithm that uses a number of iterations that is expressed polynomially (in the form n^k where n is the number of items in the input and k some integer, see Table 1.1 for an example with $k = 3$).
- $\mathcal{NP}$-Problems (non-deterministic polynomial time) are problems that can be verified polynomially. Every $\mathcal{P}$-problem is then by definition a $\mathcal{NP}$-problem.
- $\mathcal{NP}$-Hard problems are a class of problems that are "at least as hard as any $\mathcal{NP}$-problem" and can in that circumstance be used to express any $\mathcal{NP}$-problem that exists (using the the Cook-Levin Theorem).
- $\mathcal{NP}$-Complete problems designate the problems that are at the same time $\mathcal{NP}$-hard and $\mathcal{NP}$ (verified polynomially). Karp (1972) found 21 combinatorial problems that fit this definition.

Finding a polynomial algorithm that could solve a $\mathcal{NP}$-hard model would allow the resolution of any $\mathcal{NP}$-problem in polynomial time and represents the essence of the \$1 000 000-prised challenge to tackle the $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ question. In the current situation, as no solution has (yet) been found, the mathematical complexity to solve $\mathcal{NP}$-problems can require an exhaustive search (a sequential observation of each possible solution) in the worst case scenario (Weisstein, 2002).

		$n = 10$	$n = 25$	$n = 50$	$n = 100$
Polynomial	$\mathcal{O}(n^3)$	10^{-6} s	2×10^{-5} s	10^{-4} s	10^{-3} s
Exponential	$\mathcal{O}(2^n)$	10^{-6} s	3×10^{-2} s	13 d	10^{13} yr

Table 1.1.: Difference of running time between polynomial and exponential algorithms when facing an input size n on a 10^9 operations-per-second computer (Cook, 2012, p. 8).

Given that the basic examination timetabling problem (no conflicting exams) can be remapped polynomially into a $\mathcal{NP}$-hard graph colouring problem, the first is a $\mathcal{NP}$-hard problem as well (Bergmann et al., 2014, p. 84). Their verifiability makes them $\mathcal{NP}$-complete.

Cooper and Kingston (1996) proved that (examination) timetabling problems are $\mathcal{NP}$-complete using 5 different cases that are often encountered in practice:

1. When students are free to pick the courses that will compose their degree.
2. When the number of participants to a meeting is variable.
3. When there is no distinction in quality between the slots.
4. When conditions regarding the slots to which an event can be assigned exist.
5. When resources required in a triple assignment exist in limited quantities.

On top of that "basic" complexity definition, the problem complexity increases even more when a variety of constraints to be considered are sometimes contradicting each other (Qu et al., 2009, p. 56).

When benchmarking the quality of its algorithm, Carter (1986, p. 194) noticed that randomly-generated conflict matrices are harder to solve than real ones whatever the density that is used to generate them. According to him, this would mean that a first selection method is already performed "naturally" to limit the possibilities of conflict.

Implications for solving the problem

Due to its complexity, the problem is difficult to solve using exact methods even with the extensive computational power we currently have at one's disposal (Fonseca, Santos, Carrano & Stidsen, 2017, p. 28). Indeed, the search space (the set comprising all the possible assignments and solutions) grows exponentially with the number of variables that are encountered in the problem (Burke & Petrovic, 2002, p. 267). In such circumstances, the finding of an optimal solution is only possible when the problem size is small (Qu et al., 2009, p. 62).

As Oude Vrielink, Jansen, Hans and van Hillegersberg (2017, p. 4) state in their paper, "feasible, efficient, or fast solutions are all synonyms for polynomial time". As it cannot be simplified to a $\mathcal{P}$ problem, researchers often try to apply various kinds of heuristics to find solutions, even if it does not result on finding of the overall best solution. Thanks to the propriety relative to problem reformulation, any heuristic method that is used on a $\mathcal{NP}$-complete problem can be used (or at least tried with various effectiveness) to solve an other $\mathcal{NP}$-complete problem more efficiently (Kingston, 2014, p. 250).

1.3. Historical review of E.T. research

Problems relative to timetabling are studied closely by scientists originating from operational research (OR) and artificial intelligence (AI) since the years 1950's (Burke, Bykov & Petrovic, 2001, p. 118; Qu et al., 2009, p. 55).

While conceptual approaches for education have been formalised in the beginning of the 1960s, the **first modelling** truly dedicated to university examination timetabling dates from 1964 when Sol Broder from the State University of New-York tackles the problem using conflict matrix (Section 3 for definition) in order to minimise the number of conflicts by using a heuristic algorithm (Carter, 1986, p. 196). Even if the results obtained seem poor by today's standards, this was the first documented approach trying to use computer-based techniques to solve that sort of problems though.

Earliest techniques consisted merely in trying to automate the operations that the human scheduling officers were doing manually by scheduling first the examinations they qualify as the most difficult (Schaerf, 1999, p. 89). Due to the limited computational power at that time, the main focus was then to find a solution where "exams in the same period should not have common students" (Burke et al., 2004, p. 510).

In the 1980's, thanks to the **increasing access to computing facilities** in most of the academic institutions, the number of researchers approaching these problems using computer based algorithms growed (de Werra, 1985, p. 151). The trend in these years was to use individual resolution methods though justifying that because "problems can be very different between one school and another one", the building of an universal timetabling program did not seemed reasonable (de Werra, 1985, p. 161).

A review of the topic performed in Carter (1986, p. 200) highlights the **absence of communication between authors** which were often unaware of the algorithms and methods used by others. This led to a weak level of comparison between the performance of individual approaches until the apparition of benchmark data sets (Burke, Kingston & Pepper, 1998, p. 213). Due to these conditions, Carter stated that no "overall best method" could stand out at that time.

Since 1995, the Practice and Theory of Automated Timetabling (PATAT) community organises every two years an **international conference** that gathers the most regarded scientists on timetabling problems. Another popular conference, the International Timetabling Competition (ITC) is being held more sporadically and matches the same objectives. Such conferences help the scientists to stay updated on the latest trends and algorithms while opening their mind up to new horizons with new perspectives and aspects that would not have been explored otherwise (Kingston, 2014, p. 249).

The popularity of the topic has increased even more since the beginning of the 21st century. Research trends since then explore the possibilities offered by single-solution based heuristics methods and develop new forms of hybridisation of heuristic methods that could increase the "level of generality at which search methodology can operate" (Qu et al., 2009, p. 55).

1.4. Why change might be necessary?

There are 3 major reasons to be found in the literature that support the idea of using computer-assisted techniques to draft examination timetabling:

- As higher education institutions face this time-consuming problem several times a year, the development of a solution can be worth the investment regarding the time and resources that can be saved afterwards (Qu et al., 2009, p. 55).
- The employee should keep a bird's eye view of the problem at any moment to assign properly an event to the timetable. Even if doing so is possible for small cases, it gets more challenging when the number of examinations to schedule reaches the hundreds (Bergmann et al., 2014, p. 83).
- The problem tends to get even more complicated due to the increasing trend of offering cross-disciplinary curricula where students can pick a set of classes to develop skills that are not located in the core of their training (Oude Vrielink et al., 2017, p. 8). Indeed, this 'flexibility' increases the risk of mutual overlap.

Computer-based techniques, when programmed properly, are able to overcome most of the issues previously mentioned. On top of that, the use of an intelligence-free entity makes the creation of workable solutions quite fast even if the size of the problem and the number of conflicts is large (Carter, 1986, p. 193). On the other site, defining if a solution is superior to another is hard to feel for a computer and represents a task that a human being would do better (Schaerf, 1999, p. 90).

With this "philosophy" in mind, the main goal of using such techniques to draft examination timetables can be summarized as a way of facilitating their generation, increasing the efficiency of the resources required to perform the operations, a total replacement of the manually-performed operations being excluded from its immediate objective.

1.5. Elements considered in E.T. models

Before listing them, a large variety of elements can indeed be considered in examination timetabling models. Based on an exhaustive analysis, Schaerf and Di Gaspero (2007) state that "timetabling problems not only vary from country to country, but also from university to university, and even in different departments of the same university". This involves that developed systems are often only launched in the department (or institution) in which they have been designed (Bardadym, 1996, p. 22).

Under that assumption, a wide variety of models can be created based on the multiple combinations of the constraints that are possible. In this section, we will only describe the most frequent ones that are found in the scientific literature. We will specify the elements to be considered in our model later on in Section 3.2.

1.5.1. Constraints and preferences

When we review the theory, the authors tend to make a clear distinction between what they qualify as "hard constraints" and "soft constraints" (Burke & Petrovic, 2002, p. 266):

- **Hard constraints**, also referred to as first-order conflicts, designate conflicts that should be avoided by all means and cannot be violated under any circumstance (Bergmann et al., 2014, p. 84; Burke et al., 2001, p. 120). They are mostly used to determine cases where the use of particular resources (i.e., teachers, rooms or students) are mutually exclusive (Qu et al., 2009, p. 56). When a solution respecting all the hard constraints is found, it is qualified as "feasible" (see Section 1.5.2).
- **Soft constraints**, referred to as second-order conflicts as well, are used on the other side to express constraints where their fulfilment is desirable but not absolutely critical (Qu et al., 2009, p. 56). A penalty must be associated to each type of "violation" in order to quantify its wrongness, furthermore they can be weighted according to the number of persons suffering of the grievance (Carter, Laporte & Chinneck, 1994, p. 111). Soft constraints should be satisfied as far as possible but even if they are not, in opposition with hard constraints, it does not change the feasibility status of a solution (Landa Silva et al., 2004, p. 91).

Examples of hard constraints

- **Exclusive resources** should not be assigned to different events simultaneously (Burke et al., 2001, p. 120): a student can pass only one exam per period, a room can be occupied by a single exam at a time, etc.

- **Resources** should be **sufficient** but represent settings that cannot be altered without changing the definition of the problem (Qu et al., 2009, p. 56; Müller, 2016, p. 259): the exam session counts X periods, the rooms have a predetermined number of seats, maximum N exams can be organised per period, etc.
- **"Unique" hard constraints** exist as well where a Kuwaiti university for instance requires the students to be split according to their sex (McCollum, McMullan, Parkes, Burke & Qu, 2012, p. 293).

Examples of soft constraints

- The **closeness of examinations** is often taken into consideration in the generation of timetabling in order to provide enough time to study for the next exam (Bergmann et al., 2014, p. 84).
  - Laporte and Desroches (1984) introduced what became the most popular technique to quantify the dissatisfaction of 'too close exams'. They use a quadratic penalty ($p_{i,j}$) which is inversely proportional to the number of slots (up to 4 in their formulation) between the examinations i and j:

$$p_{i,j} = \begin{cases} 2^{5-|s_i - s_j|} & \text{if } 1 \leq |s_i - s_j| \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

    with s_i and s_j being the slot attributed to the exam i and j respectively.
  - Dummy slots are often used in combination to take into account the existence of larger time lapse between two slots otherwise the penalty would be identical (Carter et al., 1994, p. 113). For instance, with an exam on Saturday afternoon, the next exam on Monday morning would be considered as consecutive (+16) even if in reality there are 2 slots in-between (+4).
- Some **disgraced slots** exist where stakeholders prefer not to have exams in their planning; it includes the first and last day of the session and Saturdays (Bergmann et al., 2014, p. 88).
- The properties of each exam, room and slot are variable meaning that **some combinations are perceived as more desirable than others**: schedule large exams earlier to facilitate their marking (Qu et al., 2009, p. 56), take account of the relative difficulty of examinations (Bergmann et al., 2014, p. 88), favour the use of a single room for exams, etc.
- There are **soft constraints that can be expressed as hard constraints** as well according to the willingness to consider these event requirements (Burke & Petrovic, 2002, pp. 266–277). It includes: exams in a particular room/slot, relative position of two exams, exams that should be simultaneous/consecutive, etc.

1.5.2. Desirability of a solution

First of all, it is worth noting that a clear distinction is made between **search problems** and **optimisation problems** (Schaerf, 1999, p. 88). Indeed, search problems "merely" aim at finding a solution that fits all the hard constraints; such a solution is qualified as **feasible**. On top of that, optimisation problems aim at finding the feasible solution that violates "the least" the soft constraints; such a solution is qualified as **optimal**. Any feasible solution does not require any modification to be applied even if they do not satisfy the stakeholders (McCollum et al., 2012, p. 297).

Even if the earliest applications of examination timetabling algorithms were used to solve 'basic' search problems, current research now focuses on improving the **"desirability" of the solutions**, given that any feasible solution is not necessarily convenient to the involved parties (Bergmann et al., 2014, p. 82). The very existence of a feasible solution makes the quantification regarding their consideration of the soft constraints possible (Burke & Petrovic, 2002, p. 272).

A measure that can be used to determine the room for improvement is to compare the number of slots that are required in an unconstrained conflict-free problem (see Section 2.1.1 f.i.) and the number of slots available in the problem; the larger the difference, the more likely it is to find better solutions (Carter, 1986, p. 201). A major issue exist though regarding complexity: even if a feasible solution can be found easily, the finding of the optimal one requires much more computation (Carter et al., 1996, p. 374).

Subsections below will cover several other topics relative the desirability of a solution: the prioritisation of soft constraints, the use of sums as single measure of quality and fairness between students.

Prioritisation of soft constraints

Given that finding a single solution that can fulfil all the soft constraints at the same time is purely hypothetical, making some trade-offs regarding the fulfilment level of each type of constraint are most of the time necessary (Burke et al., 2001, p. 118). Generally speaking, the "wrongness" of a particular violation will be associated to a weight which will help to define the relative importance of a criterion with regard to another. Timetabling officers are the ones that define what weight will be associated to what particular undesirable configuration (subjectively or objectively).

On top of that, it is necessary to stay in a similar order of magnitude for the different penalties in the objective function or otherwise some would be totally ignored when the solver tries to find a solution to the problem, see Table 1.2, (Terashima-Marin, Ross & Valenzuela-Rendon, 1999, p. 604).

	Obj. fct.	Exams on Sat.	Contribution	
			Ind. pen.	Saturday
$w_2 = 0$	53 088	92	53 088 (100.0 %)	0 (0.0 %)
$w_2 = 4$	58 292	28	56 408 (96.8 %)	1 884 (3.2 %)
$w_2 = 16$	62 687	20	57 247 (91.3 %)	5 440 (8.7 %)

Table 1.2.: Impact of varying the penalty (w_2) associated to the planning of an exam on Saturday while other penalties remain identical (test on a 3600 s run on FIAL-201806 dataset).

Sums as a measure of quality

As stated in the section relative to the "desirability of a solution" (Section 1.5.2), we face two different kind of problems: a *search* or an *optimisation problem* and each prizes the quality of the solutions differently. In the first case, a potential measure of quality could be the percentage of examinations that are scheduled without conflict while, on the other side, an objective function composed of one or several measures has to be minimised or maximised (Schaerf, 1999, p. 114).

For the same author, the minimisation of a weighted sum as sole objective function lacks of a "natural interpretation" of the value meaning that, for instance, a penalty of 19 740 is definitively better than one of 27 320 but it does not describe intrinsically the improvement that has been achieved between the two. Indeed, in such case, it is necessary to analyse the whole set of data to determine its origin. On top of that, it is conceivable that a precise value of the objective function can be derived in several ways using a different assignment of the variables (Burke et al., 2001, p. 119).

The criteria, as they might be of different nature, make the contradiction with each other possible, meaning that abnormalities will be reported in the objective function (Burke et al., 2001, p. 121). For instance, the administration finds it desirable to set two exams in adjacent periods (a written and an oral part of a same exam for instance) while the model will try on its side to penalise such proximity. In that case, some adjustments are required to avoid such issues by conditioning the enforcement of a penalty for instance.

How to produce fairer solutions?

We are facing an allocation model where scarce resources (in our case rooms and slots) have to be distributed among a large set of users (the students) which induces some level of inequality among the users (Jain, Chiu & Hawe, 1984, p. 2). As mentioned earlier, most of the models in the literature use the minimisation of a single objective function (composed of weighted penalties) to define the "quality" of a solution. Muklason, Parkes,

Özcan, McCollum and McMullan (2017, p. 302) believe that the way such problems are formulated can lead to unfairness among students because minimising a sum of penalties still leaves some individuals with a higher penalty than the others, according to the configuration, even if the solution is the best one overall.

Albeit there is no absolute definition of a fair distribution examination timetabling, the key factors can be found back in the one commonly used in economics where "no person in the economy prefers anyone else's consumption bundle over its own", the allocation is then said to be 'envy-free' (Muklason et al., 2017, p. 305). Given that the study performed by the same authors tend to show that the students attach a lot of importance to how the allocation fits them, showing them that the situation is similar for all the other students could decrease their dissatisfaction.

Due to the gaps existing in the previous methods used to qualify fairness (variance, coefficient of variation and min-max), Jain et al. (1984) introduced a new methodology to define the fairness among groups. Their paper mention 4 desired properties of a fairness index:

1. Population independence: an index should be workable whatever the size of the population.
2. Scale and metric independence: the value of the index should be the same whatever the scale or unit used for the penalty/prize.
3. Boundedness: bounded values help in intuitive understanding of the index (e.g. 0 being totally unfair, 0.2 being 20% fair and 1 being totally fair).
4. Continuity: any change of attribution should have an impact on the index.

The model they developed, the Jain Fairness Index (JFI), matches these 4 requirements (Jain et al., 1984, pp. 5–6):

$$\text{Jain Fairness Index} = \frac{[\sum_{i \in I} x_i]^2}{n \sum_{i \in I} x_i^2} \tag{1.1}$$

where $(i \in I)$ designates a member of the population I, n the size of the population and x_i the penalty/prize attributed to the member i.

On top of that, the JFI still works when the attribution of resources required is asymmetrical in absolute terms (e.g one does needs more resources than an other) meaning that individual attributions (x_i) can be weighted according to their own requirements. This is a major property to exploit in examination timetabling given that people who have more examinations have higher minimal penalty level than the others (see Figure 1.1). Slight variations can be observed between the different sessions. Two factors seem able to explain these: the number of slots and the day a session starts (see

Table 5.1). A further study of these variations will be performed in the Section 5.2.2 of the second part of this work. Given that the structure of the weeks is not symmetrical, a combinatorial study of the length and the start/end days vary on too many points to study them properly on short notice.

Given that the weeks are not symmetrical, the study of

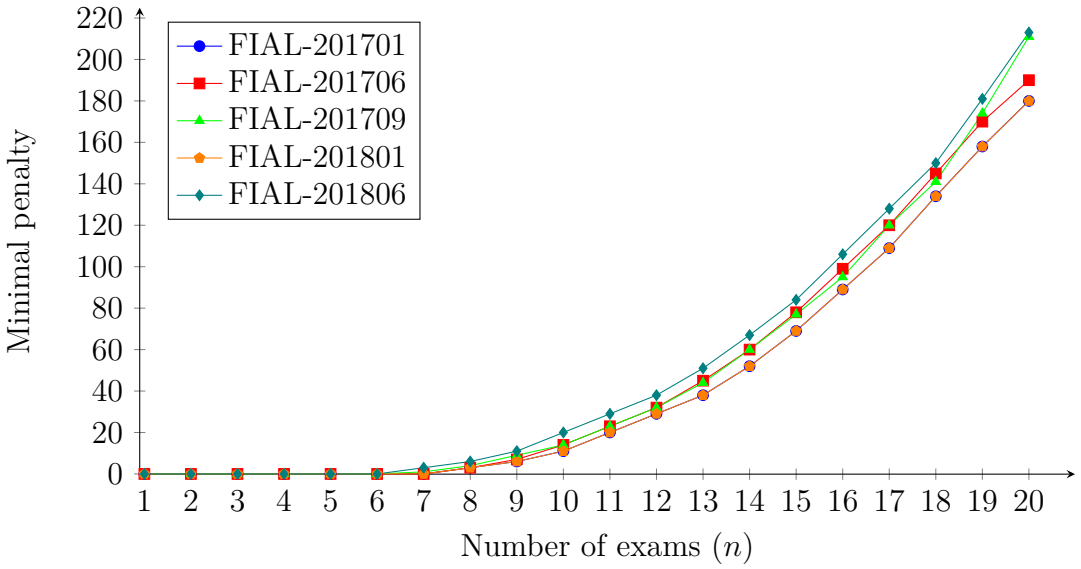


Figure 1.1.: Absolute minimal penalty for an individual who has to carry n exams in the studied configurations (2 exams a day, no exam on Sunday, penalty of 4 on Saturday) and according to Laporte and Desroches (1984) definition.

Given that the presented index used to measure the equity of a solution uses sums of quadratic powers, it increases even more the complexity of the "base" problem. Furthermore, the minimum value reached for the objective function tends to increase slightly (Muklason et al., 2017, p. 309). It implies that the decision maker has the final word on whether he prefers an absolute lower value of the objective function or a solution that would be fairer to the students.

In the study that we carry on afterwards, the index will only be used as a supplementary metric to measure and distinguish the quality of the solutions produced by the Mixed Integer Programming (MIP) model. Following that, the first objective will not be to maximise its value but to give new indications on the produced schedule and compare it to the values reached when the generation of a schedule is performed manually.

1.6. Performance of timetabling tools

Due to the size of the problem and its NP-hard complexity (see Section 1.2), the probability to find the optimal solution in a short amount of time is unlikely even with the progress of computational power that have been made since the first formulation of these problems.

In such circumstances, the main challenge lies in the finding of the appropriate equilibrium between the quality of the solution and the time required to find it. This trade-off seems logical given that, under normal conditions, a longer search would allow the exploration of a larger part of the search space, increasing the chances of finding an improved solution (Burke et al., 2004, p. 512).

As stated in the introduction, the creation of examinations timetables is performed "merely" three times a year: in Fall, Spring and Summer. Furthermore, given that the finding of an actual solution is just a step in the whole generation process, it should be made clear what is considered an acceptable run time. Indeed, when preparations take several weeks, a difference of seconds, minutes or even hours to the time required by the algorithm does not seem that significant but on the other side an algorithm taking days or weeks would not be acceptable either in the case of further alterations of the parameters being needed (Burke et al., 2004, p. 512).

1.7. Stakeholders: actors or victims?

Prida Romero (1982) states that there are three main groups of people which have to cope with the outcome of an examination timetable grid: the administration, the departments and lecturers and finally the students. According to the size of the university and the way it is administered, the first two could be encountered in a single decision-making body. This would be the case in an university where the schedules for all faculties are generated by a centralised entity. Furthermore, studies tend to show that timetable-related problems cause dissatisfaction among all the groups of stakeholders (Oude Vrielink et al., 2017, p. 2).

Administration of the university Their role is to determine the quality threshold that any examination timetable should meet. Indeed, even if computers can perform the finding of a workable solution, they are capable of defining on their own neither the resource requirements nor the direction to reach (de Werra, 1985, p. 151).

Departments/Lecturers Each department faces its own set of problem and preferences regarding how the exam sessions should be organised: relative position of an exam to another, preferential position for large exams, specific rooms requirements (e.g. chemistry lab, computer room).

The implementation of automated timetabling faced quite some resistance in the '80s among the faculties' staff given their impossibility to control the process until the computer finishes producing the timetable from the data entered (Oude Vrielink et al., 2017, p. 8). The issue has been overcome since then thanks to the development of 'interactive scheduling' in the '90s which provides the opportunity to alter a solution visually from within the software (Bardadym, 1996, p. 34). Even if timetabling software seem user-friendlier than before, most of them remained based on 'old fashioned' algorithms limiting the progress realised during the last decades (Oude Vrielink et al., 2017, p. 13).

Students Last but not least, even if they only face the result of the process, students are the primary stakeholders affected by the quality of a generated examination timetabling though. They are the ones who will have to adjust their (study) schedule according to the position of examinations, the length of the interval between two exams, the holding of a meeting in a specific part of town or even on inconvenient days. Indeed, even if the real impact of a particular variation seems quite difficult to demonstrate, a non-negligible part of the students tend to believe that their examination timetabling "affect negatively their academic achievement" (Muklason et al., 2017, p. 306).

Cowling, Kendall and Mohd Hussin (2002) wrote a review where they highlighted the preferences and critics of students using a questionnaire regarding the perception of their timetable. The most frequent claims include: the need for gaps between exams, a lower acceptance of exams on weekends and a preference for an uniform distribution of exams over the examination period. Complains formulated were mostly directed to a "lack of conscientiousness" of the faculty personnel: the late delivery of the grid, the appearing incompatibilities and the need for alterations after the first publication.

2. Solving examination timetabling problems

This chapter will give a short overview of the different methods that can be used to solve examination timetabling problems. As scientific literature does not seem to evade from trends and fashion, successive waves of heuristic methods have been observed since the 1990's. Indeed, since the beginning of research about this topic, a huge variety of methodologies (or even combinations of these) have been used more or less successfully to tackle this sort of problems.

The most common first level of separation is performed between exact and heuristic methods. The first refers to an algorithm that is able to guarantee the finding of the optimal solution. But, as the complexity increases rapidly in the case of problems with binary or integer variables, the decision maker might have to limit himself to a solution found by a heuristic algorithm that is "merely" close to the optimum. Indeed, he could decide to use heuristic solutions due to the lack of computational power or because of the worthlessness of additional efforts.

2.1. Exact methods

2.1.1. Graph colouring[1]

A **graph** $G = (V, E)$ is a way of describing a network composed of two type of elements (vertices and edges) that are contained in the two separate sets V and E (see Figure 2.1 for an example). The first, $V(G)$, includes all the vertices (also known as nodes or points) that are included in the graph while the second, $E \subseteq [V]^2$, includes all the edges that might connect these vertices. By convention, edges are noted using $\{i, j\}$ with i and j being vertices.

Two nodes i, j from the same graph G are considered **adjacent** when $\{i, j\}$ is an edge that exists in $E(G)$. A graph G is qualified as **undirected** when no distinction is made between the edges $\{i, j\}$ and $\{j, i\}$ for all edges in $E(G)$.

A **weighted graph** W designates a graph where each edge $\{i, j\} \in E(W)$ has an attributed a numerical weight $\omega_{i,j}$ that embeds a setting that is considered in the model: a distance, a capacity, a penalty, etc.

[1] Definitions to be found in this section are adapted from Diestel (2017) unless mentioned otherwise.

The **degree of a vertex** designates number of neighbours that a vertex can reach directly using the edges composing the graph; a vertex that has no neighbour is said to be **isolated**.

Vertex colouring of a graph G is the association of a colour c to each vertex in a way such that adjacent nodes do not share the same colour $c(i) \neq c(j)$ for all $\{i, j\} \in E(G)$.

The **chromatic number** of a graph, $\chi(G)$, refers to the minimal number of colours that is required to colour the vertices of a graph. A graph is said to be k-chromatic when $\chi(G) = k$ and k-colourable when $\chi(G) \leq k$.

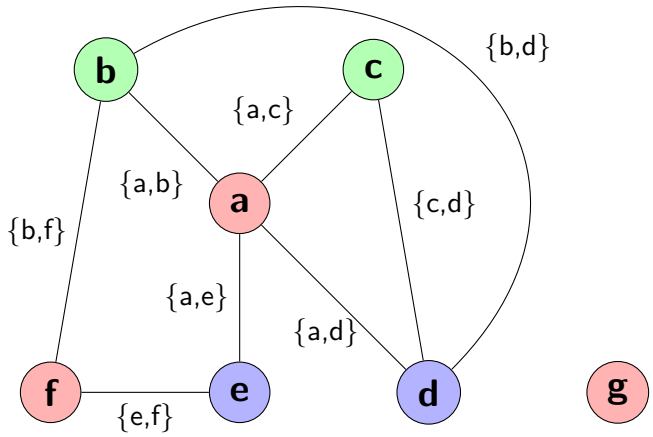


Figure 2.1.: Example of an undirected graph composed of 7 nodes $\{a \dots g\}$ and 8 edges. The graph, being colourable using 3 colours but not 2, is 3-chromatic.

Graph theory was one of the first techniques that have been applied to solve examination timetabling problems and is still used but at a lesser measure. Indeed, even if such methods are efficient for large models, some constraints (examination distance, assigning rooms, etc.) cannot be expressed using a pure graph-based formulation (de Werra, 1985, p. 151). Some other constraints can be modelled though by adding new edges that embody other types of conflicts such as external exclusion of simultaneous events.

The generation of a conflict-free schedule can be compared to a case of graph colouring where each node is an exam, edges are drawn between nodes that have at least one student in common and the objective is to find the minimal number of colours required to colour the graph in a way such that two adjacent nodes do not share the same colour (Burke & Petrovic, 2002, p. 267). Afterwards, every period will be associated to a single colour which will ensure the lack of conflicts for elements sharing the same colour. Its formulation only looks after the hard constraint where a student can pass a single exam per period. The chromatic number represents the absolute least number of slots that are required to organise a conflict-free exam session (Müller, 2016, p. 265). Nowadays,

it is mainly used as a way of finding an initial feasible solution that will be improved further using different meta-heuristic algorithms.

Figure 2.1 provides an example of a graph that could be coloured using 3 colours: $\{a, f\}$, $\{b, c\}$ and $\{d, e\}$ (g being isolated, can take any of the three colours).

Heuristic methods for graph colouring

Carter (1986) states that graph colouring is a $\mathcal{NP}$-complete problem meaning that the time required to find the optimal solution grows exponentially. To overcome such issues, even if they do not ensure optimality, heuristic methods are used to assign sequentially colours to the different vertices within a short time. The most crucial part of the algorithm is then to rank the different examinations according to a measure of difficulty that will be used while keeping in mind that assigning a slot gets more and more difficult as the number of already scheduled events increases (Burke & Petrovic, 2002, pp. 267–268; Qu & Burke, 2009, p. 1276). The most popular difficulty measures used to define the next to add to the schedule include:

- **Largest degree first**: vertices are ranked according to the number of direct conflicts they have with other vertices. When weights are used to quantify the importance of a particular conflict (relative to the number of students having to face both), it is qualified as a 'largest weighted degree'.
- **Colour degree**: vertices are ranked according to the number of conflicts that exist with elements that have already been scheduled.
- **Saturation degree**: at each step of the construction, nodes are classified according to the least number of valid periods that could host the event and their conflict degree respectively.

The last method, also known as the Brelaz's DSATUR algortihm, is the most popular of these and allows to find solutions which are close to the ones found using exact algorithms.

2.1.2. Integer programming

Integer programming formulations constitute an extension of linear programming models where some (or all) variables are meant to take integer values. The use of integer variables allows the modelling and enforcement of constraints that could not be expressed using the "simple" LP form. They are mostly used to express decisions that have to be made among a finite set of options. On top of that, binary variables constitute an even more specific type of integer variables that are restricted to the values 0 (zero/no) and 1 (one/yes).

While it is possible to solve most of the linear problems in polynomial time using algorithms such as the simplex, no computationally effective algorithm exists for solving all types of IP's. Due to that, integer programs are usually harder to solve than their linear counterparts, strongly limiting the size of the models that can be solved to optimality within a short time lapse. Integer programming models that can be solved efficiently include (Wolsey, 1998, pp. 37–52): minimum-cost flows problems, shortest path, maximum weight tree, etc.

A generic formulation of an integer programming model can be found below:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} && (2.1)\\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} && (2.2)\\ & \mathbf{x} \geq 0 && (2.3)\\ & \mathbf{x} \in \mathbb{Z}^n && (2.4) \end{aligned}$$

Let's review the different components of this formulation: $\mathbf{c}$ refers to the vector $(n \times 1)$ of coefficients that will be multiplied in the objective function by the value of the variables; $\mathbf{x}$ is the vector $(n \times 1)$ of variables (integer in this case); $\mathbf{A}$ is the matrix $(m \times n)$ of coefficient of the constraints and $\mathbf{b}$ is the vector $(m \times 1)$ of the upper bounds of the constraints.

On the other side, as mentioned previously in the section relative to problem complexity, as the possibilities of assignment grow exponentially with the number of variables, the resolution of IP problems is still computationally expensive even if some methods such as *branch and bound* allow a faster search (Qu et al., 2009, p. 62).

Another non-negligible aspect of using IP formulations is that it provides the possibility to calculate a lower bound for the objective function of the problem allowing to estimate the quality of the solution that has been found until that moment (Fonseca et al., 2017, p. 28). Otherwise, even if the computer has been running for several days, it still would not be possible to define whether the solution found is close to optimality or not (Bergmann et al., 2014, p. 96).

Two of the most important arguments in favour of integer programming techniques are the ease to translate an idea into a proper formulation and its huge flexibility regarding the relations and constraints that can be expressed, which is crucial for some problems (Qu et al., 2009, p. 62).

More specifically, when translating the creation of an examination timetabling problem into integer programming, it can be expressed as a problem where we try to assign values (binary or within a range) to a set of variables (relative to different associations

and parameters) in a way such that it satisfies the constraints while limiting secondary violations (Burke et al., 2001, p. 119).

Special ordered sets

Special Ordered Sets have been introduced by Beale and Tomlin in 1973. IBM CPLEX documentation defines them as "an additional way to specify integrality conditions in a model" by limiting for each set the number of variables that can be non-zero. There are two types of SOS:

- Special ordered sets of Type 1 (aka. SOS1) are sets of variables where at most one variable from the set may be non-zero while all the others are equal to zero (Beale & Forrest, 1976, p. 53). This situation is often encountered in integer programs where we have to settle at most on a single choice among all the possibilities.

$$\sum_{i \in I} x_i \leq 1$$

- Special ordered sets of Type 2 (aka. SOS2) are sets of variables where at most two of these may be non-zero. If there are two non-zero variables, they must be adjacent in the set as well (Beale & Forrest, 1976, p. 53).

SOS are useful in branch-and-bound methods to branch directly on sets of variables instead of individual variables. Doing the branching based on sets will accelerate the exploration of the search space.

Transposition of the graph colouring problem into IP

The previously mentioned high flexibility of integer programming techniques allows the translation of the minimalist "conflict-free graph colouring" problem used to find the chromatic number of a graph which is the most basic formulation of an examination timetabling problem. Such formulation allows to make use of the advantages provided by the different IP solvers.

- **Sets**
  - I Set of all exams to schedule
  - P Set of periods of the exams session
- **Indices**
  - i, j Indices for exams, $(i, j) \in I$
  - t Index for periods, $t \in \{1, \ldots, P\}$
- **Parameters**

$c_{i,j}$ Equal to 1 if at least one student is enrolled for both exams i and j, 0 otherwise

N Number of exams in the session

- **Decision variables**

$$x_t = \begin{cases} 1 & \text{if slot } t \text{ is used} \\ 0 & \text{if not} \end{cases}$$

$$y_{i,t} = \begin{cases} 1 & \text{if the exam } i \text{ is scheduled in slot } t \\ 0 & \text{if not} \end{cases}$$

$$\text{minimize} \quad \sum_{t=1}^{T} x_t \tag{2.5}$$

$$\text{subject to} \quad \sum_{i \in I} y_{i,t} \leq x_t \cdot N \qquad \forall t \in T \tag{2.6}$$

$$\sum_{t \in T} y_{i,t} = 1 \qquad \forall i \in I \tag{2.7}$$

$$c_{i,j}(y_{i,t} + y_{j,t}) \leq 1 \qquad \forall (i,j) \in I, \forall t \in T, i \neq j \tag{2.8}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{2.9}$$

$$y_{i,t} \in \{0,1\} \qquad \forall i \in I, \forall t \in T \tag{2.10}$$

The objective function is to minimise the number of slots that are being used during the session while respecting three constraints: the first (2.2) requires the activation of the period to assign exams to it, the second (2.3) forces all exams to be assigned to a single period and the third (2.4) allows only the scheduling of a single exam when they are conflicting ($c_{i,j} = 1$). The two last (2.5 and 2.6) define the domain of the variables x_i and $y_{i,t}$ that need to be binary.

This formulation will be used afterwards to calculate the minimal number of slots that is required to create an examination timetable. The model will be extended using some other parameters and variables to take into account the difference between major and minor slots for the exams.

2.2. Heuristic methods

Heuristics have been used for a long time to cope with $\mathcal{NP}$-problems even if the terminology appeared much later. Heuristic methods differ from the exact ones by their ability to provide "good solutions" within a short lapse of time even though they are neither capable of guaranteeing the optimality nor of providing an optimality gap, the distance of a solution to a lower bound (Wren, 1996, p. 46).

Such methodologies are based on a process of performing iterations to inspect the neighbourhood using different techniques to find a solution with a better fitness until a predetermined stop-criterion is met (Burke et al., 2004, p. 510).

The diagram presented below (Figure 2.2), serves as a **categorisation** of all the meta-heuristic methods that are commonly encountered to cope with difficult problems. Even if the diagram suggests an exhaustive categorisation of the different meta-heuristic methods, the scientific literature on examination timetabling problems limits itself to two categories: single-based approaches and population-based approaches (Oude Vrielink et al., 2017, p. 6). We will follow the flow of the literature by using the two-way classification.

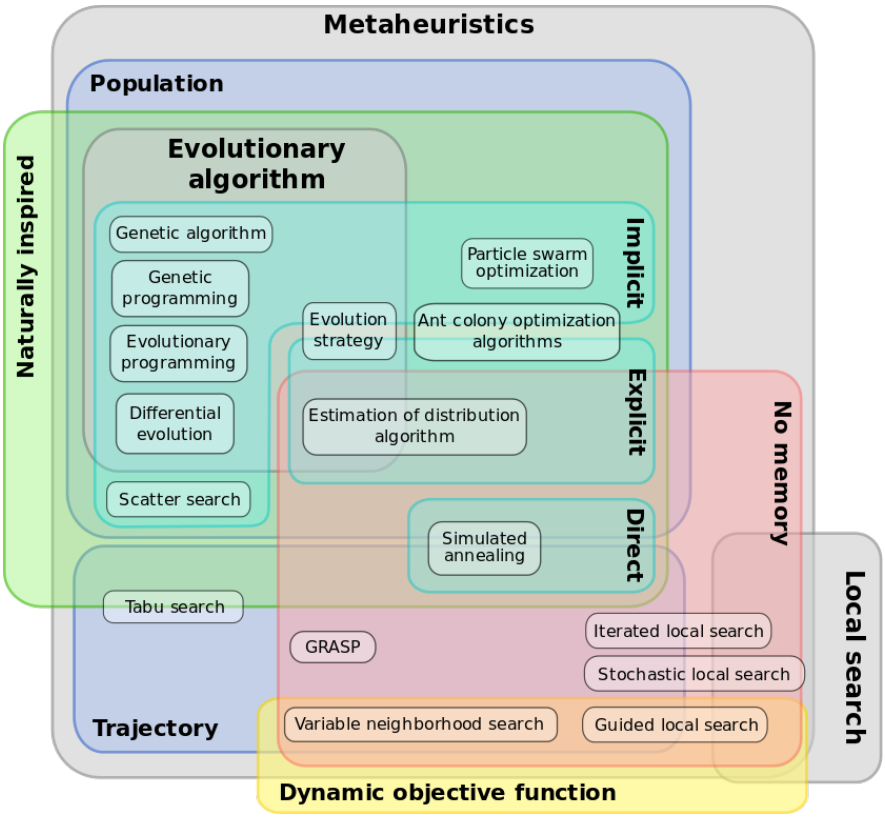


Figure 2.2.: Euler diagram of the different types of meta-heuristics (Wikipedia).

According to the same author, each category provides a particular advantage regarding the improvement that they can bring: the single-based are said to *exploit* the search space while the population-based *explore* it. By 'exploiting', they mean finding better solutions within the neighbouring ones (using different search techniques) while 'exploring' aims at producing new, improved, solutions based on the most interesting existing ones (constituting the population). Population-based techniques face a major disadvantage though: the more the algorithm progresses, the least diversity remains among the population, meaning that it would converge quickly with a limited exploration unless curative methods are being used (Oude Vrielink et al., 2017, p. 7).

None of the two categories manages to dominate examination timetabling problems given that the search space to be explored is massive and the topography of solutions hilly: solutions are "disjoint from each other" and the local extrema are numerous (Bardadym, 1996, p. 33).

Another issue for meta-heuristic techniques lies in the large efforts that are required to *tweak* the model in order to reach competitive solutions (Qu et al., 2009, p. 65). Indeed, according to them, an identically configured meta-heuristic algorithm could reach very different levels of quality when tested on different datasets. Qu et al. (2009, p. 69) believe that the need for experimental approaches in such methods makes its use by timetabling officers difficult to implement. To overcome that problem, hyper-heuristic methods operate at a higher level of abstraction which will pick the best heuristic method to be applied at each iteration.

2.2.1. Single-solution based meta-heuristics

As said previously, single-based heuristics are techniques that are trying to solve problems "by searching from an incumbent solution to its neighbourhood" (Qu et al., 2009, p. 63).

In this subsection, we will cover the three most popular methodologies used in the scientific literature to build examination timetables. These are: Tabu search, Simulated annealing and the Great Deluge.

Tabu search

Proposed by Glover in 1986, Tabu search is different from generic hill climbing techniques on two factors: (1) moving to a worse solution is accepted when no improvement is possible (when arriving at a local extremum); (2) the successive addition of the previously accepted solutions (or moves) to a list which embeds all the solutions where it is not possible to go back to in further iterations (Wren, 1996, p. 61; Qu et al., 2009, p. 63).

The "tabu list" prevents the solver from going back to previously visited solutions allowing the solver to escape from the local search space and extremum (Burke et al., 1997, p. 7). The limitation of the 'length' of the tabu list allows to come back to a previously found solution after a predefined number of iterations in order to relax the criterion (Burke et al., 2004, p. 511).

Simulated Annealing

Simulated annealing is a heuristic method that is similar to the concept of "annealing" used in metallurgy since the 1950s; its main concept lies in the control of the cooling of the material to improve the quality of crystallisation and alter the proprieties of the

material compared to a naturally cooled situation. As the cooling is controlled, the element gets smoothly to the lowest energy level which is analogous to the minimisation of a function.

The first step consists in mapping the neighbourhood of the current solution to define the possibilities among which a move will have to be picked randomly at each iteration. The move is directly accepted if it improves the solution found so far and according to a probability related to an iteratively decreasing temperature if not. As the temperature decreases successively, the *chance* to pick a worse solution diminishes leaving only improved solutions when arriving close to the "freezing point" (Burke et al., 1997, p. 7; Wren, 1996, pp. 60–62).

Great Deluge algorithm

The Great Deluge (GD) shares the same "philosophy" that accepts worse solutions under some conditions. The algorithm works with an "impassable" upper/lower bound (B) that is multiplied by a decay-rate/growth-rate (ΔL, see Figure 2.3) which will tighten the zone of accepted new solutions at each iteration (Burke et al., 2004, p. 513). For a minimisation, a neighbouring solution below B will be accepted but strictly refused if above B. A previously found feasible solution is used most of the time as the initial bound.

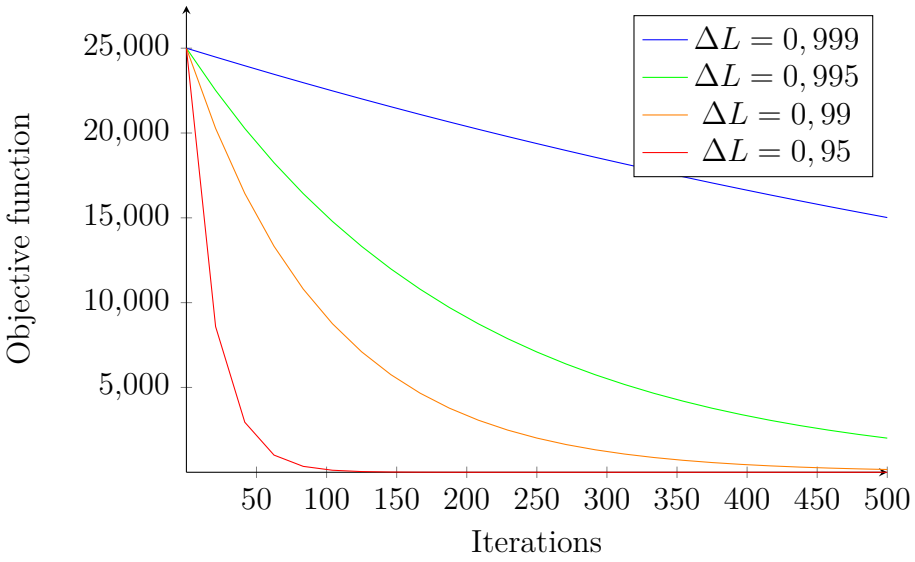


Figure 2.3.: Influence through iterations of four different decay-rates (0,999, 0,995, 0,99 & 0,95) on the evolution of an upper-bound (B) starting at 25 000.

Its denomination relates to the need for finding a higher elevation (an improved solution) during a flood with increasing water level (B) without getting wet (Qu et al., 2009, p. 64). Once the water reaches our level and no higher place can be reached, we are at a local/global optimum.

2.2.2. Population-based meta-heuristics

Population-based meta-heuristics, in opposition with local-based search, require the existence of a population of solutions that will be exploited iteratively to generate new and (hopefully) improved solutions. Such algorithms cannot be applied on any population though: some variety is required among the starting solutions to avoid a quick termination due to a lack of diversity (Burke & Petrovic, 2002, p. 171). In subsections below, two major evolutionary algorithms (*Genetic* and *Memetic*) and one swarm algorithm (*Ant colony*) will be briefly described.

Genetic algorithm

The Genetic algorithm (GA), the most studied of the evolutionary algorithms, makes a clear parallel with *Darwin's Evolutionary Theory* where the process of selection is based on the well-known "survival of the fittest" phrase. It embodies that, in a competitive world, only the most desirable features are kept to sustain the next generations (Burke et al., 1997, p. 6). The information, being coded as *chromosomes*, will be transferred to the next generation using a transmission process that uses crossover (picking selectively some information from a specific parent) and mutations (alterations) with the aim of getting solutions of increased quality at each iteration (Burke & Petrovic, 2002, p. 270; Qu et al., 2009, p. 66).

Even if their application on existing solutions is conceivable and has already been done previously, Qu et al. (2009, p. 66) state that they are better in the role of a selection of the algorithms in a hyper-heuristic (see Section 2.2.3) environment.

Memetic algorithm

The Memetic algorithm (MA) is an extension of the Genetic algorithm where a local search is applied on each new individual to improve the quality of the members of the population before the "creation" of the next generation by the GA (Qu et al., 2009, p. 67). The main idea behind this local search aims at constituting a population which, being solely composed of local optima, would facilitate the constitution of further improved solutions (Burke et al., 1997, p. 6; Burke & Petrovic, 2002, p. 270).

It can be considered as a form of hybridation between global and local search techniques. This would combine the advantages of both the exploration done using population-based methods and the exploitation done using local search techniques (Oude Vrielink et al., 2017, p. 6). Being composed of two phases, it is difficult to determine and balance the time to dedicate to each one. On top of that, the performance regarding optimality is variable from dataset to dataset and lacks of consistency (Burke & Petrovic, 2002, p. 269).

Ant colony optimization

This algorithm is based on the functioning of a swarm of ants which individually leave pheromones behind when walking to keep track of the route they travelled. Pheromones being volatile, only the most recent tracks (*in extenso* the shortest ones) remain detectable for the ants and, if no-one goes through that route for a while, it will completely lose its attractiveness and simply disappear (Wren, 1996, p. 62). Starting from random moves, this "social method" allows to keep the most desirable properties at each iteration and use it to build new, improved, solutions (Qu et al., 2009, p. 67). On the other side, the stronger is the "smell", the least likely they will diverge to find another route limiting themselves to the best ones found so far.

2.2.3. Hyper-heuristics

Hyper-heuristics is defined by Burke and Petrovic (2002, p. 278) as a methodology that uses a "heuristic to choose heuristics". Indeed, as stated before, the parameters of meta-heuristic algorithms need to be tuned to make them work on the particular instance of a problem. Qu et al. (2009, p. 69) even considers it "as being as difficult as that of developing new approaches". The objective of hyper-heuristics is to overcome this issue by "operating at a higher level of abstraction" than the previously mentioned techniques (Oude Vrielink et al., 2017, p. 7).

As implied by the "naïve definition", it is performed in two layers with the low level heuristics (that are used to improve the current solution) and the high level heuristics atop (that are used to pick the 'best' heuristic among the low level heuristics) (Qu & Burke, 2009, p. 1274). In an ideal world, being separated from a specific case, its "modular architecture" allows its use on problems with different characteristics without requiring any "expert intervention" for adjustments (Muklason et al., 2017, p. 304).

2.3. Comparison of techniques

In each situation, picking the most appropriate solution to solve this type of problems is quite difficult and experimental as no unique comparison factor exists to determine it. The reasons are multiple:

First, each time that authors imagine and try different methodologies, they usually experiment them solely on their own set of data meaning that, while it works in a particular situation, it is not a proof that the savings are reproducible on other datasets. Such lacks do not help distinguishing a particular approach that would be superior to than another (Schaerf & Di Gaspero, 2007, p. 40).

Second, in the real world, many criteria have to be considered simultaneously to gauge the capacity and the quality of a provided solution other than the pure "performance" of the algorithm (Schaerf & Di Gaspero, 2007, p. 43). Such criteria could include: the objective function, the gap reached, the time necessary for a solution, the ease of the software maintenance, its interoperability with other programs, the presentation of the outputs, the training needed by operators, etc.

Given our limited skills in programming, we set our choice on a proven solution that we already used before: an integer programming formulation implemented using AIMMS language. This formula seemed the most appropriate for our experimentations as our input will be solely limited to the formulation of the model and the testing conditions. On top of that, the software does not require any other manipulation than the data entry avoiding the need for complex tweaking of the algorithm on a "trial and error" basis.

II

Practical application within the FIAL faculty

In this part, we will cover the practical aspects of conceiving an examination timetabling model that could be used as a tool afterwards to generate examination timetables for future sessions in the specific context of the FIAL faculty.

This part, after a brief introduction about the faculty, counts three different chapters:

- In the first, we will expose the different key points relative to the integer programming model we conceived including the necessary simplifications, the requirements and an exhaustive formulation of the model in mathematical notation.
- Based on the format introduced in the previous chapter, we will speak about the required data preparations in order to use it as an input to be solved by our model. We describe afterwards some points about its application in AIMMS, the modelling software that we used, such as: the implementation, the data input and extraction and the solver that we used.
- Last but not least, we perform a testing of our model on five previous cases: January 2017, June 2017, September 2017, January 2018 and June 2018. We compare the results obtained with the ones from manually-generated solutions. We will study then the variation of several criteria and their impact on the quality of the solutions. Finally, based on our observations, we will share some ideas that we hope, could deserve to be studied in the future.

As stated in the introduction (and even the title), the main application of this thesis is to implement mathematical techniques to facilitate the production of examination timetables for the FIAL faculty of the *Université catholique de Louvain* (UCL). "FIAL" is a portmanteau built on the French name of several different disciplines that are taught there: *Faculté de philosophie, arts et lettres* (lit. Faculty of Philosophy, Arts and Letters).

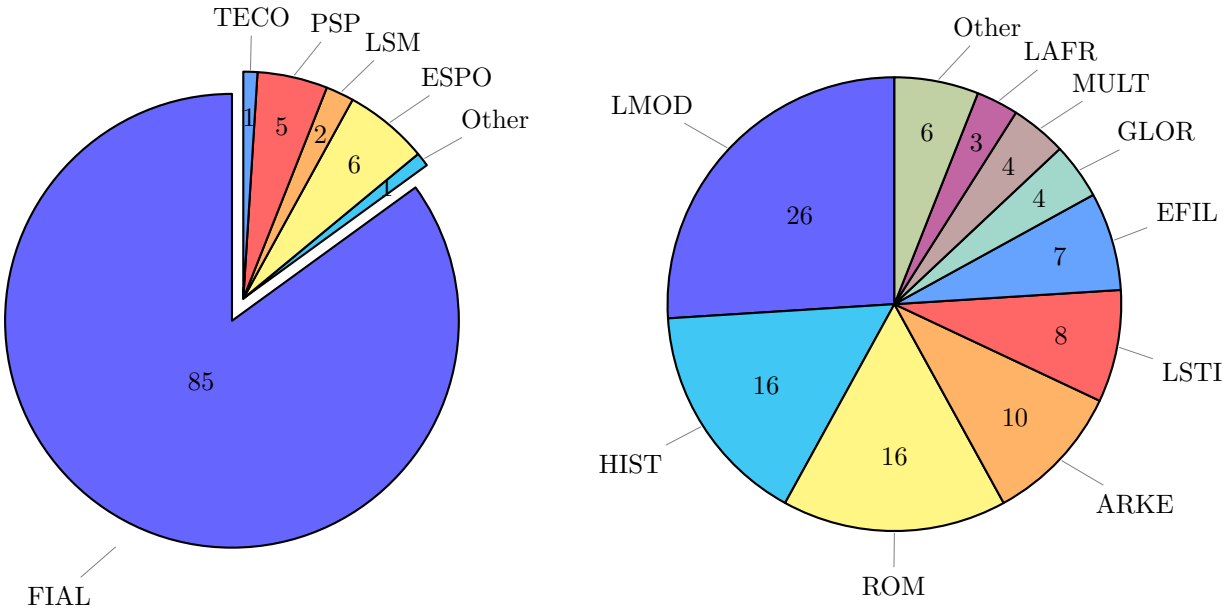
The degree courses offered at FIAL stretch from the Bachelor to the Master degree (and an optional doctorate afterwards) unlike the *Louvain School of Management* faculty (LSM) in Louvain-la-Neuve who only offers Master degrees and requires the student to already hold a bachelor from another faculty (most of the students originate from ESPO programmes though).

The most popular studies include: Languages and Literatures (either Ancient and Modern; and either Romance, Germanic or Oriental), History, Philosophy and History of Art. On top of this variety of programmes, students in Bachelor are encouraged to "broaden their perspective" by choosing a minor (30 ECTS out of 180) which is different from their core classes (from Law to Economy, including Musicology, Sociology or even Management). As some of these classes are not organised by the FIAL faculty itself, it needs to be taken care of when building the timetable to avoid conflicts due to external circumstances.

Some attempts of adjustments seem to have been made to avoid such conflicts by defining a set of slots, around the end of the session, where all the minors exams should be scheduled in an ideal world. For various reasons, this does not seem to be applied methodically though as some of these minor examinations are still planned outside this subset and they represent on average merely 20 to 25 % of all the exams scheduled in these slots.

Figure 2.4a below shows an approximate breakdown of the students having to pass at least one FIAL exam according to the faculty they belong to. The data has been extracted only from the January and June datasets given the unreliable nature of September session being a resit. It is very clear, with around 85 %, that the biggest share of the students originates from FIAL faculty, while 'visitor students' share the remaining 15 %. There is no particular variation from year 2017 to year 2018 for each faculty except a slight increase at the LSM where a new policy requires students to take a mandatory language class as a substitute for exchange programmes.

Among the 85 % previously mentioned, a second breakdown has been done (see Figure 2.4b) to have a precise overview of the popularity of the different categories of curricula



(a) Students breakdown according to the faculty they belong to (in percent).

(b) FIAL students breakdown according to the curriculum they follow (in percent).

Figure 2.4.: External and internal breakdown of students passing FIAL examinations (in percent).

proposed at FIAL faculty. The list below provides information on the most popular ones as of 2018, ranked by the number of students:

1. **LMOD** (26 %): Contemporary Romance and Germanic languages and literatures;
2. **HIST** (16 %): History;
3. **ROM** (16 %): French and Romance languages and literatures;
4. **ARKE** (10 %): History of Art and Archaeology;
5. **LSTI** (8 %): Translation and Interpreting;
6. **EFIL** (7 %): School of Philosophy;
7. **GLOR** (4 %): Ancient languages and literature;
8. **MULT** (4 %): Multilingual communication;
9. **LAFR** (3 %): Ancient and contemporary languages and literatures.

Visual representations of the five different examination sessions have been joined in the Appendices. Being built as a graphs, each vertex represents a direct conflict between a pair of exams that have to be scheduled during the same session. All the nodes have been coloured according to the slot attributed by the Brelaz's graph colouring heuristic. After a clustering, most of the courses belonging to a same curriculum will be found in a specific region of the graph. Courses in the middle of a region have stronger links with the other members of this region than if they were in the periphery where we meet mostly courses stretching out on multiple ones.

3. Model definition

After reviewing in the first part the theory relative to examination timetabling and methods that can be used to solve this type of problems, we opted for an Integer Programming formulation to express the requirements and constraints before implementing the model in AIMMS, a modelling environment that serves as an intermediary between the model and the different types of solvers available (see Chapter 4 for more details).

In this chapter, we will define the different ins and outs of the model that will be implemented in the next phase. It will start with a set of simplifications used to streamline the formulation of the model. It will be followed then by an exhaustive review of the requirements that it must fulfil. After that, all the different components of the IP formulation will be covered and explained to clarify their individual role in the formulation.

3.1. Simplifications

Due to the problem complexity in its 'complete form', several adjustments have been made to the model with the view of avoiding to face unfeasible problems while keeping a consistent representation of the problem. Such simplifications include:

- Regarding the **timeslots**:
  - 2 slots per day: a first in the morning and a second in the afternoon;
  - Being each up to 6 hours long (8–14 and 14–20), they should theoretically fit any exam;
  - Weekend: two slots on Saturdays but no slot on Sundays. The use of Saturday slots is slightly depreciated though.
- **Oral examinations** require some adjustments to transform them into a special case of a written exam. These adjustments include:
  - The time required per student/group defines the number of students that can pass the exam during a slot.
  - From that observation, an oral examination will be partitioned in several sub-exams if required, allowing each sub-exam to be attributed to its own time slot. The model must enforce that none of these sub-exams can happen simultaneously (risks of teacher conflicts otherwise).
  - When a partition is performed, each student will be attributed the sole sub-exam that concerns him.
  - Oral examinations that have already been attributed a slot are hard-constrained in the model before the computation is launched.

- **Special forms of examination** such as exams for evening classes are not taken care of.
- The model applies to the "classic" **three-sessions-a-year-scheme**: mid-term exams are not considered.

3.2. Elements taken into account

On an internal basis, the major key points and requirements relative to the course of an examination session are controlled by internal regulations embedded into the Academic and Examination regulations (in French, *Règlement général des études et des examens*) which are approved yearly by the Academic Council. These regulations include:

- Each course is evaluated using either an oral examination, a written one, a continuous evaluation, a work or a combination of the previously mentioned (Art. 60, 61).
- There are three exam sessions a year, one after each term (*quadrimestre*): January, June and September sessions (Art. 65).
- The examinations are scheduled irrespective of their "theoretical term". There are some restrictions though regarding when a student is able to take a resit exam:
  - An exam failed in January or June can only be resat in September of the same year (except for first year students who have three opportunities to sit their January exams);
  - There are only resit exams during September's session.
- The begin and end dates of each session (and thus their length) are decided on and published before the start of the academic year in the Academic Calendar. Unless exception, any exam should be scheduled within these dates (Art. 66).
- The definitive examination timetable has to be delivered to the students 10 days before the start of the session at the very latest (Art. 75). Alterations are forbidden after that, except in case of absolute necessity (Art. 87).
- When creating the timetable, the faculty staff must look after the possibility of passing all the examinations that a student registered for (Art. 80, §5).
- Adjustments for religious, philosophical or ideological reasons are neither allowed nor considered when creating the schedule (Art. 80, §6).
- Examinations cannot take place on Sundays and on public holiday (Art. 89) and they must take place between 8 in the morning and 8 in the evening (Art. 90).
- For each written exam, at least one full professor (failing that, its supply) needs to be available (Art. 92). He can rotate between the locations when the exam needs to be split in several rooms though. For oral exams, its presence is mandatory (Art. 93).

In summary, regarding the constraints that we skimmed through in Subsection 1.5.1, the model that we will build will enforce:

- **Hard constraints**
  - No student should ever have to pass two different exams simultaneously. In the same vein, professors should avoid any conflict: strictly for oral exams, but loosely for written exams.
  - No examination will be carried on Sundays and on public holiday.
  - By default, we try to respect the official length and number of slots of the examination session. When there is no other way around, some slots can be added but they should be penalized heavily to avoid unnecessary assignments.
  - Respect the special requirements of some exams: pre-assignment, availability, consecutivity and (non-)simultaneity.
- **Soft constraints**
  - The main objective will be to limit the discomfort that can be caused by an inappropriate assignation of examinations. Two major elements are considered: the relative distance between exams and the use of least favourable days.

On top of that, in the current situation, the secretarial staff tries to assign the exams for first and third year students in the morning while "dedicating" the afternoon to the second year exams in order to limit the conflict possibilities. While this technique sounds good when the timetable generation has to be done manually, Carter et al. (1994, p. 114) observed that it increases the problem's complexity and decreases the quality of the generated solution. This additional constraint will not be enforced in our model but could constitute another point to study about examination timetables.

Regarding the minor exams, the office mentioned that they should be planned ideally in the last five days of the session even if it is not enforced by a specific rule. When we analyse the existing datasets, we can observe that between 80% and 90% of these are actually taking place in these 10 slots. But, as they represent "merely" one third of the registrations, no restriction seems to exist regarding the organisation of major exams in these slots.

3.3. Complete formulation

This section mentions all the different components of the IP (sets, indices, variables, parameters and constraints) and how they contribute to the finding of a feasible solution.

Sets (and subsets)

T Set of time slots

- T_{Sat} Set of Saturday slots, $T_{Sat} \subseteq T$
- T_{Una} Set of unavailable slots, $T_{Una} \subseteq T$
- T_{Min} Set of minor slots, $T_{Min} \subseteq T$
- T_{Maj} Set of major slots, $T_{Maj} \subseteq T$
- T_{Odd} Set of odd slots, $T_{Odd} \coloneqq \{t \in T \mid t\%2 = 1\}$

E Set of examinations

- E_{Min} Set of minor examinations, $E_{Min} \subseteq E$
- E_{Maj} Set of major examinations, $E_{Maj} \subseteq E$
- E_{S_s} Set of examinations to be taken by the student s, $E_{S_s} \coloneqq \{i \in E \mid e_{s,i} = 1\}$
- E_{K_i} Set of examinations that are conflicting with examination i, $E_{K_i} \coloneqq \left\{ j \in E \mid \sum_{s \in S} e_{s,i} e_{s,j} > 0 \wedge i \neq j \right\}$

G Set of gaps between two examinations that should be penalized, $G \coloneqq 1, \ldots, \Gamma$

S Set of students

Indices

t Index for time slots, $t \in \{1, \ldots, \Upsilon\}$

i, j Indices for exams, $(i, j) \in E$

g Index for time gaps, $g \in \{1, \ldots, \Gamma\}$

s Index for students, $s \in S$

Parameters

$e_{s,i}$ Student enrolment matrix

$$e_{s,i} = \begin{cases} 1 & \text{if the student } s \text{ has enrolled for the (sub-)exam } i \\ 0 & \text{otherwise} \end{cases}$$

$a_{i,t}$ Availability matrix

$$a_{i,t} = \begin{cases} 1 & \text{if the exam } i \text{ can be scheduled in } t\text{, by default} \\ 0 & \text{otherwise} \end{cases}$$

$f_{i,t}$ Fixation matrix

$$f_{i,t} = \begin{cases} 1 & \text{if the exam } i \text{ must be scheduled in } t \\ 0 & \text{otherwise, by default} \end{cases}$$

$c_{i,j}$ Consecutivity matrix

$$c_{i,j} = \begin{cases} 1 & \text{if the exam } j \text{ should be scheduled directly after the exam } i\text{, on the same day} \\ 0 & \text{otherwise, by default} \end{cases}$$

$m_{i,j}$ Simultaneity matrix

$$m_{i,j} = \begin{cases} 1 & \text{if the exams } i \text{ and } j \text{ must take place simultaneously} \\ 0 & \text{otherwise, by default} \end{cases}$$

$n_{i,j}$ Non-simultaneity matrix

$$n_{i,j} = \begin{cases} 1 & \text{if the exams } i \text{ and } j \text{ cannot take place simultaneously} \\ 0 & \text{otherwise, by default} \end{cases}$$

Υ Total number of slots (Sundays included) of the session, $\Upsilon \in \mathbb{N}^*$

Γ Maximal distance between two examinations that should be penalized

Penalties

p_g Penalty for events being separated by a gap of $(g-1)$ slots

$$p_g = \begin{cases} 2^{(\Gamma - g)} & \text{if } 1 \leq g \leq \Gamma \\ 0 & \text{otherwise} \end{cases}$$

ψ Penalty for events occurring in depreciated slots (f.i. $t \in T_{Sat}$)

Decision variable

$$x_{i,t} = \begin{cases} 1 & \text{if the exam } i \text{ is scheduled in the slot } t \\ 0 & \text{otherwise} \end{cases}$$

Basically, only one type of decision has to be made when we compose an examination timetable: assign the different exams to the available periods while respecting the constraints that are inherent to the definition and requirements of the problem. The assignation is here enforced using a binary variable ($x_{i,t}$) that determine whether an exam is scheduled in a particular period. This variable, to be found in many of the constraints, will serve as a base as well to define the values of the two following deviational variables.

Deviational variables

$$y_{i,j,g} = \begin{cases} 1 & \text{if the gap between the exams } i \text{ and } j \text{ is equal to } (g-1) \text{ slots} \\ 0 & \text{otherwise} \end{cases}$$

From the previously mentioned decision variable $x_{i,t}$, the deviational variable $y_{i,j,g}$ is calculated to define the temporal distance between the two exams i and j. It is used afterwards in the formula below to assign a penalty when this pair of exams is "merely" spaced by $(g-1)$ slots. Given that penalties are only attributed when there is a conflict between a pair, it is not necessary to calculate it for pairs of examinations that are non-conflicting or to far from each other according to the standard, lightening the computation required.

$$p_s = \sum_{i \in E_{S_s}} \sum_{\substack{j \in E_{S_s} \\ i<j}} \sum_{g \in G} y_{i,j,g} p_g (1 - c_{i,j}) + \sum_{i \in E_{S_s}} \sum_{t \in T_{Sat}} \psi x_{i,t}$$

The individual penalty (p_s) of a student s consists in the sum of each penalty due to the relative closeness of a pair of exams that the student has to pass and the scheduling of some in undesirable slots (here on Sundays). Penalties are aggregated individually per student in order to facilitate the comparison of the penalty involved for each solution and compare it to the theoretical minimum in the optimal configuration in order to calculate the Jain Fairness Index (JFI).

Objective function and constraints

Having defined the required sets, indices, parameters and variables, the integer programming model used to solve the exam timetabling problem can be formulated as follows:

$$\text{minimize} \quad \sum_{s \in S} p_s \tag{3.1}$$

$$\text{subject to} \quad \sum_{t \in T} x_{i,t} = 1 \qquad \forall i \in E \tag{3.2}$$

$$\sum_{t \in T_{Min}} x_{i,t} = 1 \qquad \forall i \in E_{Min} \tag{3.3}$$

$$x_{i,t} + x_{j,t} \leq 1 \qquad \forall i \in E, \forall j \in E_{K_i}, \forall t \in T \tag{3.4}$$

$$x_{i,t} + x_{j,t-g} + x_{j,t+g} - y_{i,j,g} \leq 1 \qquad \forall i \in E, \forall j \in E_{K_i}, \forall t \in T, \forall g \in G \tag{3.5}$$

$$f_{i,t} \leq x_{i,t} \leq a_{i,t} \qquad \forall i \in E, \forall t \in T \tag{3.6}$$

$$x_{i,t} = x_{j,t+1} \qquad \forall (i,j) \in E, \forall t \in T_{Odd}, \text{when } c_{i,j} = 1 \tag{3.7}$$

$$x_{i,t} = x_{j,t} \qquad \forall (i,j) \in E, \forall t \in T, \text{when } m_{i,j} = 1 \tag{3.8}$$

$$x_{i,t} \neq x_{j,t} \qquad \forall (i,j) \in E, \forall t \in T, \text{when } n_{i,j} = 1 \tag{3.9}$$

$$x_{i,t} = 0 \qquad \forall i \in E, \forall t \in T_{Una} \tag{3.10}$$

$$x_{i,t} \in \{0,1\} \qquad \forall i \in E, \forall t \in T \tag{3.11}$$

$$y_{i,j,g} \in \{0,1\} \qquad \forall (i,j) \in E, \forall g \in G \tag{3.12}$$

First and second constraints (3.2 and 3.3 respectively) are used to force the assignation of each examination (or sub-event) to a single time slot. The second, completing the first, requires that minor exams are scheduled specifically in their dedicated set of slots. Constraint 3.4 prevents that two conflicting exams are scheduled simultaneously.

Constraint 3.5 is used to determine if the distance between each pair of conflicting exams is located among the range of gaps that are to be penalized. If the exam i and j are separated by $\pm g$, two of the three binary decision variables are equal to 1 requiring that the deviational variable $y_{i,j,g}$ takes 1 to make the sum stay below 1.

Constraint 3.6 enforces two things at once: that pre-assigned exams must happen in the predetermined time slots (using $f_{i,t} = 1$) and that exams must be attributed among slots where it is possible to hold them (for various reasons stated hereafter). The second parameter ($a_{i,t}$) allows the expression of other types of unavailable resources than the sole "conflicting exams for students" such as: when, for a factual reason, a teacher is not available in a particular slot (or a set of them); a room with special equipment is already attributed to another faculty; etc.

The next one (3.7) is required to force two exams to be consecutive on the same day. It can be activated in circumstances like when the professor wants students to pass both the written and the oral part of an exam on the same day or if he decides to dedicate a whole day for the passing of oral examinations for instance. t has to be odd given that the first slot of the day is odd in a 2 exams a day configuration. Given that it is hard constrained, the adjacent pair will not be penalized in the objective function.

The next pair of constraints (3.8 and 3.9) are used to impose that two exams happen simultaneously ($m_{i,j} = 1$) or non-simultaneously ($n_{i,j} = 1$). The first will mostly be used in the case of an exam that has to be carried between different groups (exc. people who took it as a minor) to avoid the requirement of writing different sets of questionnaires for the same course (or closely similar). The second, on its side, will mostly be used to express that a single teacher cannot be at several places at once to hold different exams. To carry on what we said in the 'Elements taken into account', it will be enforced strictly when the pair involves at least one oral exam but loosely, according to their preferences, when both are written exams.

Constraint 3.10 makes impossible the organisation of any exam in the time slots that belong to days that are forbidden with regards to the regulations into effect (Sundays and public holidays). One could wonder why we keep "impossible slots" into the model, the explanation lies in fact in the difference of distance between the two cases.

The last two require that both the variables $x_{i,t}$ and $y_{i,j,g}$ take binary values as it is impossible to partly attribute an event to a slot or to say that a distance between two exams is partly exact.

Given the flexibility of the formulation, it is totally conceivable (and easy) to add new requirements to fulfil on top of the "basic ones". For instance, even if we will not go deeper into this thesis, we successfully managed to extend it to include the assignation of rooms to the examinations. Indeed, implementing it merely required the addition of two variables, some parameters and constraints to ensure a feasible assignation of rooms to exams respecting: the attribution exclusivity, the limit of rooms for an exam and a number of seats that exceeds the number of students. The last is required because some exams involve more students than the number of seats available in a single room. In such circumstances, it is necessary to split the cohort in several rooms which is similar to the packing problem but with variable bin sizes in this case. Experimentations were done on a smaller dataset without reaching optimality meaning that it is very unlikely to be applied on large datasets without requiring (even more) computational power than the "simple one" we described in this section.

4. Model implementation

The previous chapter determined the different sets and parameters that the model is built on and that is going to be applied on the different input datasets in order to build a feasible and desirable examination timetable. Now that we know the ins and outs of the integer programming model, the first section of this chapter will cover how the data should be cleaned before, in a second step, loading it into AIMMS and running the solver.

4.1. Data preparation

One must take into account that our model tests were performed on already existing timetables. Starting from this existing information, we adapted it in order to fit it in the required input format. On the other hand, starting from scratch involves other data treatments such as the expression and propagation of conflicts. The subsections below will cover both cases.

4.1.1. Starting from existing data

As mentioned above, starting from the individual timetables requires some "reverse-engineering" to find back some of the original data in order to use it as an input of the model. Indeed, as it is built to be disseminated among all the faculty students, the information carried is limited to what might be interesting to them. Therefore, some data preparation has to be carried before building the sets and parameters of the cases. It includes:

- Removing events relative to classes that are not evaluated using exams *per se*: works and continuous evaluation (in French, *travaux* et *évaluation continue*);
- Removing exams to be taken by FIAL students that are organized by other faculties. Actually, they could be kept and constrained to happen in a specific slot but the adherence to a faculty was impossible to determine easily;
- Taking into account the different events that has to be programmed for an exam (written part and/or oral part);
- Creating duplicate events for a class when it is required (minor exams, several slots for oral examinations, etc.) and add incompatibilities between these events.

After that cleaning, the different sets can be constituted quickly using some Excel Power Query scripts. Indeed, as a language oriented to data treatments, it can perform easily operations on large datasets such as: extracting data from columns, transforming data, building new columns, removing duplicates, pivoting columns, etc.

In our case, as the teacher availabilities and pre-attributed exams stay unknown after our data exploration, these parameters remain empty during our tests. Sole the exams consecutivity could be easily guessed for different sessions organised for a single course (a course with an oral exam after the written exam or a whole day of oral exams for instance) but others would be harder to read. Mutual exclusions (used to avoid that a professor has to carry several exams simultaneously) were hard to define as well except for multiple slots required for a single course. Indeed, some of the courses are co-taught by different teachers and it is not really clear whose presence is mandatory at each event.

According to the faculty employee, these availabilities and requirements information can be extracted from the ADE Expert schedule management software meaning that it could be used in the second case presented below.

4.1.2. Starting from scratch

When the model shall be used to build new instances of examinations timetables, the data will obviously be obtained differently. As for now, the staff takes around 15 days after the closing date for inscription to collect the data from different sources and 15 more days after that to build the examination grille before communicating it to the students.

A major problem lies in the lack of communication between the different software solutions that are used in the whole university. For instance, student register for their examinations on a platform that has been made internally at the UCL. Since iCampus has been superseded by Moodle in the Academic Year 2016-2017, most of the registrations for oral exams happen through this last allowing the staff to extract the relevant information (while requiring some cleaning though). After that, the professors (un)availabilities are to be found in a private version of the ADE Expert planning tool. The last solution, which is already being used to assign courses and to deliver course schedules to students, is in theory capable of building examination timetables but is not being used for that task due to the mentioned communication issues. Other forms of conflicts, which are more rare, are entered manually by the timetabling officer.

The structure of the different data inputs that they currently use to create timetables manually is very close to the data format we require as an input of our model. In such circumstances, a different methodology to pre-process the data could suffice to use them in our model.

As the faculty staff has access to all the required data and that these sets are necessarily related somehow, all the treatments required to make the data fit in our model should be relatively basic. Given that, these operations could remain mostly hand-made or

alternatively, the development of an applet or an intelligent spreadsheet (using Power Query or macros) is conceivable as well in order to earn some time.

4.2. Model application in AIMMS

The vast majority of elements in this section originate from Bisschop and Roelofs (2004) who describe concisely the AIMMS modelling language.

AIMMS is an integrated development environment (IDE) that is used mostly to build decision tools based on optimisation models. It uses a programming language specific to the application to define all the model requirements and specifications. In opposition to most of the competing languages, the vast majority of the operations on the model can be done from a graphical user interface: building and maintaining the model, delivery of the solution results, etc.

Another point of interest is that AIMMS works in three layers: the input data, the model and the solvers respectively. Someone that performs experimentations on new data does not necessarily need to be fluent in the other layers and can limit himself to the parts that matter to him. In such circumstances, continuous improvements can be made on the model or the solvers transparently without the end-user noticing, leaving him with the best tools at any time.

4.2.1. Implementation

We used the AIMMS Developer GUI to implement the complete form of our model. The operation is very easy once the last has been laid on paper. Indeed, it merely consists in translating the different elements that constitute our model: sets (and indices), variables, parameters and constraints. Given the declarative nature of the language, once all these elements have been created, all the relations that exist between them have to be declared. The software manages without any issue all the indices related to multidimensional variables, parameters and constraints.

The language used in the software is approachable, even for novices, for relatively basic models such as ours. As more specific properties need to be used, the AIMMS manual is a good starting point for searching answers to additional requirements.

4.2.2. Data input

AIMMS is capable of reading and processing data from different sources such as text files, Excel spreadsheets, databases or other structured documents. The software is capable of processing these datasets as well to restructure the information and build dynamically new tables and sets.

Even if we mostly used direct text input in the definitions of the sets and parameters, we used extensively the processing capability though to build sets, conflict matrices, parameters tables dynamically. This is helpful to avoid having to resort to laborious manipulations in Excel when the operations could be defined programmatically in the developed model once and for all.

4.2.3. Solvers

The AIMMS academic licence, by default, offers 3 solvers that can be used on Mixed-Integer Programming models: CBC, CPLEX and XA. All solvers are not equal though: one originates from the open-source world (COIN-OR CBC) while the two others have been developed by commercial entities (IBM CPLEX and Sunset Software XA). As the last two are closed-source, each can exploit and deliver value on "secret improvements" that each made into their algorithms (or pre-treatments) making them more competitive speed-wise on certain types of problems. Gurobi is another commercial solver. On the other side, the use of commercial solutions involve expensive costs for deployment and support. A marketing presentation made by Gurobi, *Migrating an existing model to Gurobi Optimizer* (2012), shows that their deployment costs for a typical single quad-core CPU configuration are in the surroundings of $24 000 with a yearly support cost of $4800 (while IBM CPLEX asks $72 240 and $14 448 respectively).

The different solvers are compared regularly performance-wise by the Prof. Hans D. Mittelmann from Arizona State University where he uses the latest version of the solver to solve different benchmark case sets. These benchmarks exist for the most popular categories of problems: linear problem (LP), mixed-integer programming (MIP), mixed-integer quadratic programming (MIQP), etc. Using the same configuration for all runs, it determines which is the best solvers for each category of problems based on: its ability to find the optimal solution within a time lapse and the time required to find this optimal solution. In its latest review, for MIP problems such as studied here, Gurobi revealed to be best, followed by CPLEX taking between 20 and 50 % percent more time and CBC around 30 times slower (Mittelmann, 2018).

Since March 2016, AIMMS does not offer the Gurobi solver any more in their licensing package but the modelling tool is still able to communicate with it if an external licence is activated on the computer. As a result, IBM's CPLEX is now used by default in AIMMS to overcome MIP problems. For academic research, Gurobi still provides the opportunity to use their product for free, thanks to an Academic Licence programme.

Even if AIMMS picks CPLEX 12.8.0 by default, we performed all our tests on Gurobi 8.0.1 given that CPLEX behaves awkwardly. Indeed, it is able to find a feasible solution that is fair very quickly but it stays stuck at this same solution even with some additional

time (from 30 minutes to 12 hours). On top of that, its update refresh delay being very long, means that it is almost impossible to attempt to terminate earlier without killing the whole AIMMS process involving a complete loss of the results. On the other hand, Gurobi seems more stable and shows its progress regularly.

All the experiments discussed in the next chapter were performed on a personal computer (PC) with an Intel Core i5-4690k CPU running at 4.7 Ghz on four cores (and four threads), with 16 GB of RAM and with Windows 10 Pro as the operating system. The model was developed in AIMMS Developer v4.53.1 and all the tests were run again in AIMMS Developer v4.57.1 including IBM CPLEX 12.8.0 and Gurobi 8.0 solvers. Both solvers were used with their default parameters to solve the mathematical programming models. In AIMMS, a run time limit is the sole parameter that has been altered in order to avoid having to wait until the solver finds the optimal value.

5. Testing on FIAL datasets

At the early stage, this thesis was meant to assist the LSM faculty staff to produce improved and conflict-free timetables. As the person in charge there never deigned to clear up the details required to guide the research or to provide data to work on, other sources had to be found to pursue the work. Our choice fell on the FIAL individual timetables that are generated by the FIAL faculty's staff before each session and are available for the involved parties. Even if it is a synthetic table, representing the result of the assignation, they are crammed with information that are interesting for our type of research and could serve as a healthy base for our study.

As the different instances of an examination period (January, June and September) are quite similar from one year to another, we decided to limit ourselves to the study of the two last years only. In that case, the experimentation will be performed on 5 different datasets given that September 2018 was not available at the time of writing. The Table 5.1 below shows the main time-related information of the model. These concern the whole university as they are part of the Academic Calendar. Two key points can be extracted from reading this table: there is no particular pattern regarding the day of the week when they should start and June sessions are longer than the January and September ones.

	Dates		Timeslots	
	Start	End	Available	Total
201701	Fri 06/01/2017	Fri 27/01/2017	38	44
201706	Tue 06/06/2017	Fri 30/06/2017	36 (44)	42 (50)
201709	Wed 16/08/2017	Tue 05/09/2017	36	42
201801	Fri 05/01/2018	Sat 27/01/2018	38 (40)	44 (46)
201806	Mon 04/06/2018	Sat 30/06/2018	34 (48)	40 (54)

Table 5.1.: Parameters relative to the duration of the studied examination instances. As the faculty does not necessarily use all days available for the session, there are sometimes two values for the number of timeslots: the first is the one really used while the second, in parentheses, represent the maximal number of slots in theory. We used the first one in our model experimentations.

This increase of duration of June exams was required, according to the staff faculty, due to a higher number of conflicts that occur during this session. It is true that first year students, as they are able to resit their first semester exams at the same time than their second semester exams, can increase the probability of conflict between exams. This hypothesis seems to be verified when we compare the values that takes the conflict density at each instance (see Table 5.2).

The conflict density is a measure frequently used to determine or compare the difficulty involved with the creation of a specific timetable. Some authors share the idea that a higher conflict density makes the problem harder to solve. It is computed by averaging for all exams the proportion of conflicts that each exam faces (Carter et al., 1996, p. 376):

$$\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\beta_{i,j}}{N^2} \qquad \forall\,(i,j) \in E \wedge (i \neq j) \tag{5.1}$$

where $\beta_{i,j}$ is a binary value that states if exams i and j have common students (1 if yes, 0 otherwise) and N is the number of exams. Higher values show that, on average, an exam is more likely to conflict with other ones.

Table 5.2 provides interesting information relative to the studied examination instances.

	Number of			Conflict	
	Events	Students	Registrations	Edges	Density
201701	472 *(284)*	2 107	8 831	4 675 *(2 932)*	0,042 *(0,073)*
201706	516 *(311)*	1 991	10 047	6 646 *(4 333)*	0,050 *(0,090)*
201709	485 *(403)*	1 050	4 043	3 793 *(3 274)*	0,032 *(0,039)*
201801	466 *(270)*	2 118	8 545	4 568 *(2 917)*	0,042 *(0,080)*
201806	542 *(325)*	2 012	10 314	7 601 *(4 903)*	0,052 *(0,093)*

Table 5.2.: Examination-related variables of the studied FIAL examination instances. The numbers in parenthesis refer to a model being built without any duplication of the events (on par with the literature).

The columns 'Number of events', 'Conflict edges' and 'Conflict density' have each two different figures according to what is considered: the first constitutes the case when secondary factors (oral exams that require several slots and the distinction between major and minor exams) are taken care of while the second, in parenthesis, only consider the case of distinct events (each class and their written and oral variants). This distinction helps to determine if the adjustments currently performed are consistent with the requirements and the complexity of the problem.

The decrease of the density in the extended formulation can be explained by two factors: (1) As each individual can only participate to a single sub-instance of each examination, it involves that they will not grieve from the conflicts that could be introduced by people taking the exams in another slot; (2) As the number of individual events increases and given that new conflicts cannot arise, the chance of conflicts decreases by definition (or stays the same in the worst case scenario).

The conflict densities seem to correlate with the information collected during the appointments with the staff members about the relative complexity of the exam sessions. They indeed confirmed that June's sessions are harder to conceive due to the fact that some students can resit their failed January exams on top of those from the second semester even if, in absolute terms, September's sessions count more exams to schedule.

It is important to mention though that all the courses taught at the FIAL faculty are not equal when we look at the number of curricula that have the obligation/opportunity to pick them. Indeed, while some of these can only be taken under very specific conditions, some others, around a fifth of them, can be taken by 6 curricula or more. The last group is mostly composed of classes that are cross-disciplinary or can be taken as a minor. The highest number observed for a single class was 16. As they have to fit in multiple programmes simultaneously, they are harder to set in the schedule unless they are taken care of separately (in a minor week for instance) or anticipatively.

Starting from the previously stated observations, we found interesting to determine what is the absolute minimal number of slots that are required for each session only, taking care of student conflicts and also the number of slots required to pass the minor exams.

Values have been found using the integer programming formulation of the graph colouring problem that we described in Section 2.1.2, extended to take care of minor exams that have to happen in specific slots. The number of minor slots is determined by heavily penalizing the use of a minor slot (100 times the use of normal slots). From that, as the total number of slots than can be activated is limited to 40 in our experimentation, when the upper and lower bound are in the same hundred, the minimum number of minor slots can be determined precisely (always verified in our datasets). After that, the total number of slots is derived from the two last digits.

Table 5.3 below presents the results based on a 1200 seconds run using Gurobi.

	Min. number of slots	
	Total	o/w Minor
201701	19	6
201706	26	8
201709	20	8
201801	23	5
201806	28	7

Table 5.3.: Minimal number of slots that are needed to carry the session without any student conflict.

5.1. Results

Figure 5.1 shows the evolution of the objective function through time as the solver progresses in its search. As a first measure of comparison, we provided the "score" that is reached when applying the penalty function to the attribution that has been made manually by the faculty staff. It seems clear that the combination of an integer programming model with a commercial solver is technically able to reach feasible and improved solutions on short notice.

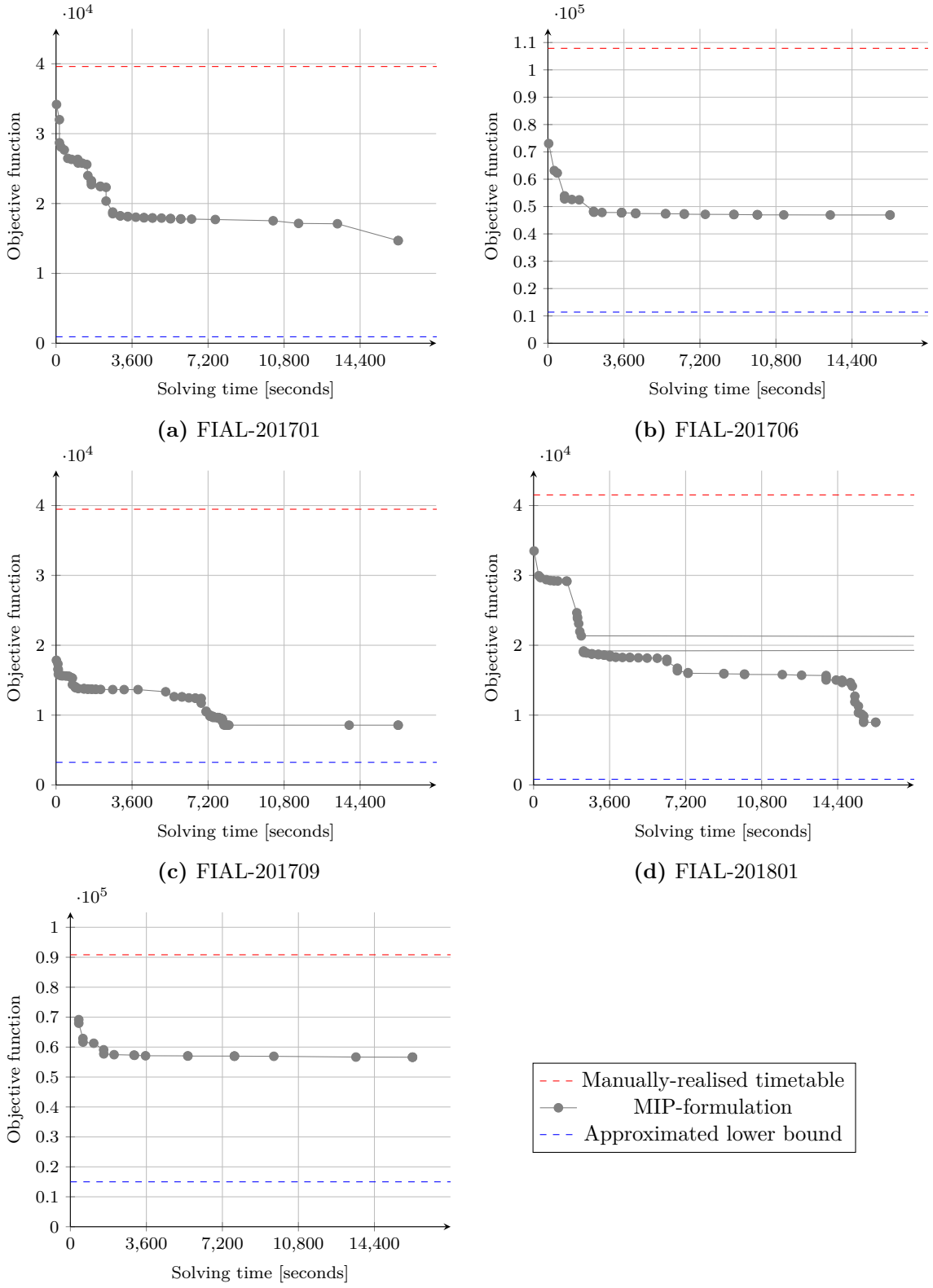
It should be noted though that due to the context of our experiment where we used existing final timetables as a base, we had to recover some information and process them before use in our model or, in other cases, we could not take them into account as it was not recoverable. Reckoning these in could decrease the benefit relative to the objective function introduced by the use of our model. These unrecoverable parameters include:

- Teacher (un)availabilities;
- Teacher carrying several exams (partially);
- Simultaneous exams;
- Minor exams outside the dedicated subset;
- Pre-determined/Forbidden slots for exams.

In the other direction, as neither of the solvers was able to determine a lower bound for the objective function during the whole run, we calculated one based on the individual penalties involved when a student has a certain number of examinations (see Figure 1.1). As these individual penalties were determined in a best case scenario situation, with a single student having to pass n exams without sharing its conflicts with other students, when these are cumulated and the optimisation is done on the whole, it is mathematically impossible to reach a lower individual penalty. One case is omitted in the minimum penalty calculation: when two exams are to take place in consecutive slots but merely 1 % of students are concerned by this measure on average.

As visible on the different sub-figures from Figure 5.2, the exams are slightly better distributed around when the model runs longer as the standard deviations decrease slightly (in an ideal world with perfectly distributed exams the SD should reach 0). It is impossible to distribute the exams perfectly unless you set it as the main goal of the optimisation as simply the use of penalized slots favour a partial distribution.

A typical week is composed of three main components: (1) The weekends are very hollow with penalized Saturdays and forbidden Sundays; (2) The days adjacent to the weekend are the most popular ones to conduct an exam facilitated by the 4 slots space in-between; (3) Small recesses can be observed in the middle of some weeks.



(a) FIAL-201701

(b) FIAL-201706

(c) FIAL-201709

(d) FIAL-201801

(e) FIAL-201806

Figure 5.1.: Progressive improvement of the objective function of the different cases studied through the increase of the solving time in GUROBI 8.0.

Given that the rooms stayed out of the scope of our model, it was necessary to make sure that no slot in the distribution was particularly overcrowded making its transposition in reality impossible. As we can see, in none of the five sets the number of simultaneous examinations exceeded 25 which seems reasonable and on par with the levels reached in manual timetables.

Even if a solution with a lower value for the objective function is better in absolute terms than one with a higher one (in the case of a minimization), we found interesting to determine whether the formulation we used improves or worsens the inequalities between students as they seem to prefer a fair competition with their cohort. To review that, we use the Jain Fairness Index (JFI) based on the values of the individual penalties of the students (p_s).

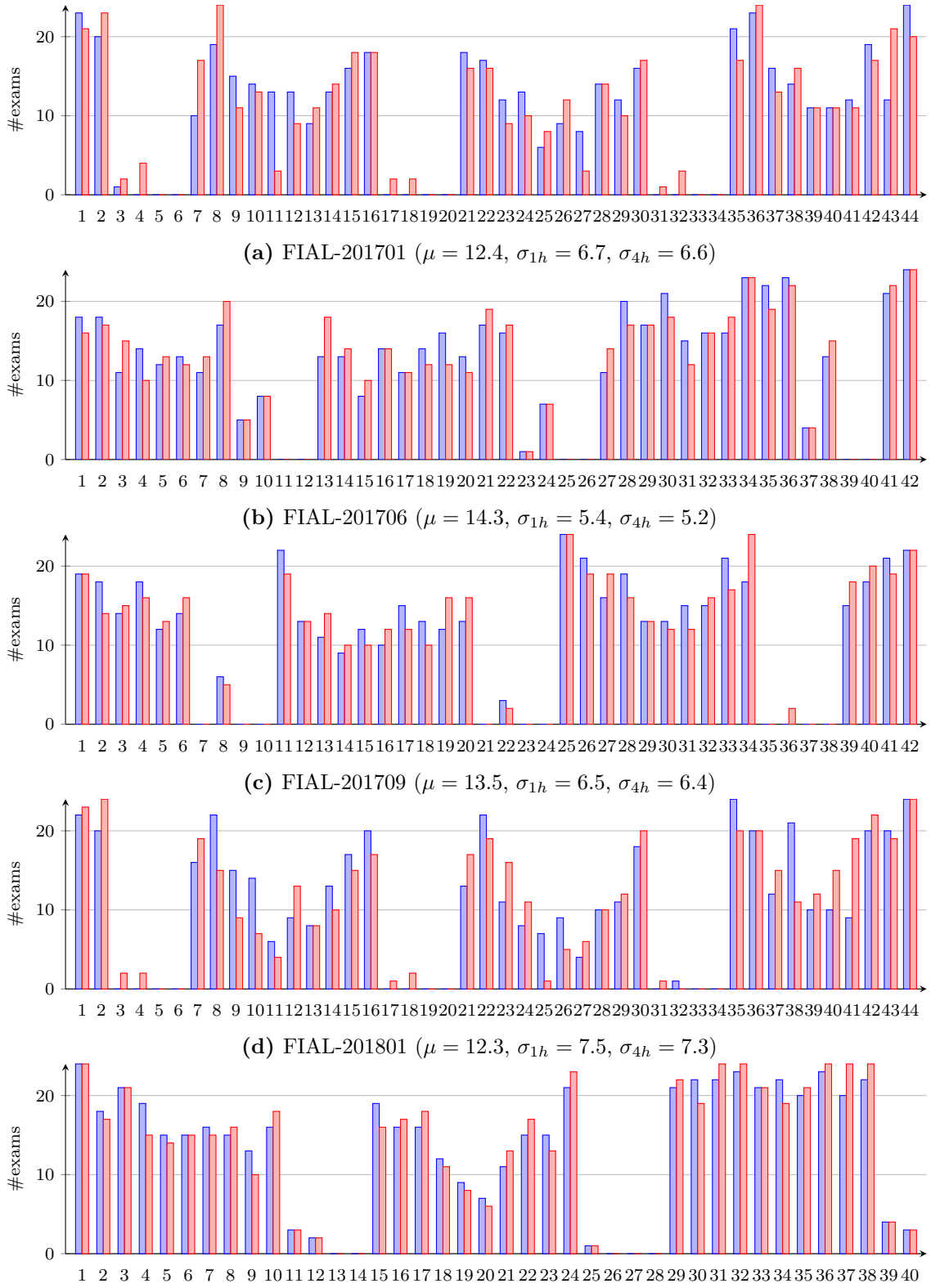
Before calculating the index, we must correct the penalties in two ways: adjust them by subtracting the minimal penalty when a student has to pass n exams and weight them according to the same number of exams. Mathematically, it can be expressed as such:

$$p_{s,corr} = \frac{p_s - MinPenalty(n)}{n}$$

We perform this correction based on two arguments: (1) we want to consider the overhead penalties only and not the ones that are inherent to the number of examinations a student has to pass; (2) the more a student has examinations, the more likely he will face overhead penalties.

The Table 5.4 shows the different values of the JFI according to the method used (by hand or model-based) and in the last case the solving time as it might have an impact as well on the index. As a remainder, the closer the JFI is from 1, the fairer the solution will be. From the values, we can observe that even with the massive reduction of the objective function, the solution does not improves the case of some individuals but of the whole population similarly (in cases where the value stays stable) or even improves the distribution of penalties (in cases where the value increases).

As both the objective function decreases and the JFI stays stable or improves, it seems undoubtedly that the quality of solutions increases for the students when our model is being used to build examination timetables. On the other hand, we must keep in mind that we left aside some of the requirements that we could not recover from the individual schedules. At least, we know that in a favourable case scenario, the “quality” for the students is not worse than it was before.



(a) FIAL-201701 ($\mu = 12.4$, $\sigma_{1h} = 6.7$, $\sigma_{4h} = 6.6$)

(b) FIAL-201706 ($\mu = 14.3$, $\sigma_{1h} = 5.4$, $\sigma_{4h} = 5.2$)

(c) FIAL-201709 ($\mu = 13.5$, $\sigma_{1h} = 6.5$, $\sigma_{4h} = 6.4$)

(d) FIAL-201801 ($\mu = 12.3$, $\sigma_{1h} = 7.5$, $\sigma_{4h} = 7.3$)

(e) FIAL-201806 ($\mu = 15.1$, $\sigma_{1h} = 7.0$, $\sigma_{4h} = 7.4$)

Figure 5.2.: Overview of the exams distributions between the slots and their change according to two different solving time durations (Blue: 1h and Red: 4h).

	Man-made	Model-based	
		1h	4h
201701	0,624	0,660	0,694
201706	0,549	0,551	0,564
201709	0,606	0,704	0,742
201801	0,609	0,693	0,802
201806	0,555	0,538	0,547

Table 5.4.: Evolution of the Jain's Fairness Index (JFI) according to the method and time used for timetable generation.

5.2. Impact of different alterations on the objective function

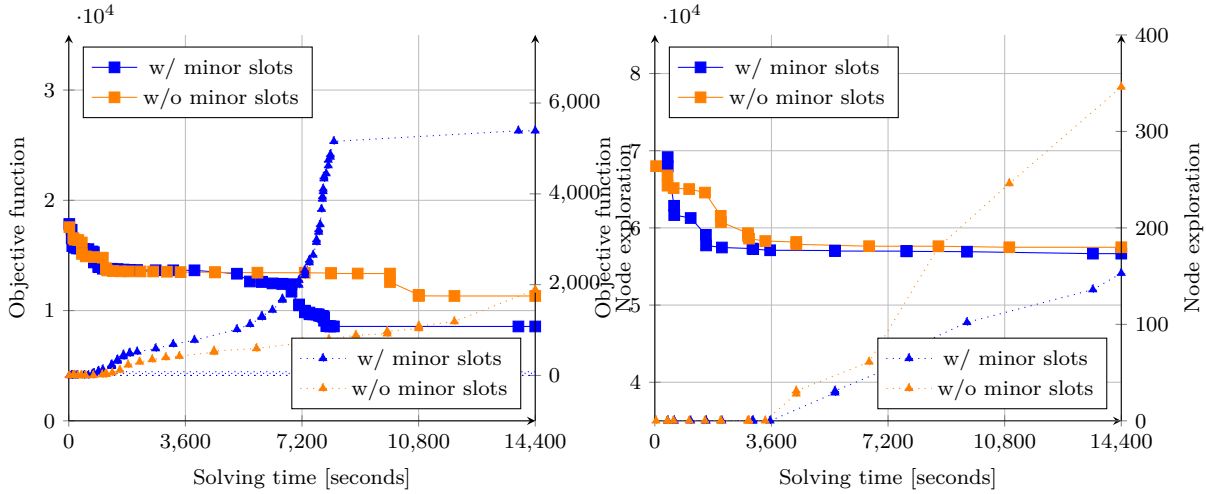
Every dataset that we studied earlier varied on multiple points, we found interesting to determine if we could extract any improvement from the alteration of some of these parameters. This section will principally focus on three types of alterations: the suppression of minor slots, the change of the start day and the extension/shortening of the duration of the session. The last two originate from the variation of the minimal penalty that can be observed in Figure 1.1. The "Minimal Penalty Model" that we used to find these values counts three points of variation: the number of exams to pass (cannot vary *per se* in real life), the total length of the session and the slots configuration (start day and number of Sundays).

In order to perform the experimentations efficiently, we had to pick a dataset which showed major improvements in a short time lapse. The examination session which seemed the most appropriate for such requirements was the "FIAL-201806" as it involves a large amount of students, and more registrations and conflicts than all the others.

As the faculty staff usually needs to be able to alter the input parameters at the last minute, it did seem interesting to study these alterations on a short lapse of time (4 hours) rather than to the optimal solution of the problem. This choice has a major impact on the interpretation of the results below. Indeed, even if the model is structured identically, applying the same alterations on different datasets can provide sometimes opposite results.

5.2.1. Suppression of minor slots

The problem definition states that minor exams are meant to happen within a predetermined range of slots, usually during a five days lapse in the last week of the session. As these examination can only be within a restricted range, we were wondering if it has



(a) FIAL-201709 **(b)** FIAL-201806

Figure 5.3.: Comparison of the objective function in cases with and without minor slots through the solving time. Plain series represent the objective function while the dotted ones represent the number of nodes explored (on the right scale).

a positive or a negative impact on the value of the objective function reached by the solver.

In order to verify this question, we tested this hypothesis of a worse solution on two different datasets: FIAL-201709 (an easy one) and FIAL-201806 (a hard one). To do so, some minor transformations had to be done on the dataset to transfer the enrolment from the minor exams to the "common ones" (excl. for the oral exams where the merge was unclear). As these occurrences are being merged, the total number of events decreases slightly.

As visible on the Figure 5.3 above, based on a 4 hours run (14 400 s), it is hard to define whether the removal of minor slots has a positive impact. Even if the two curves stay very close in the early stages of the resolution, we can see that from the moment the solver starts, the objective function seems to stay higher when no slots are dedicated to minor exams.

Given the short lapse of time on which the comparison was made, we managed to find two plausible explanations for the lack of improvement of the objective function:

1. As the number of minor-related enrolments is very limited (from 4 % to 7 %), the impact of enforcing slots for these events is very limited on the objective function.

2. As we said, we merged the written minor exams with their major counterparts. For oral examinations, as the number of compatible slots increases, the search space becomes larger meaning that the solver might require more time to perform its exploration. While the number of nodes explored through time is higher

with the minor slots on the FIAL-201709 dataset, supporting our hypothesis, the situation is reversed on the FIAL-201806 one (see series with triangular marks).

We must be cautious about these experimentations as we never managed to reach optimality meaning that the result could differ completely if the exploration was done completely.

5.2.2. Change of the duration of the session

As we mentioned earlier in the problem scope, the start and end days for the sessions are pre-determined by the academic council before the beginning of the year. In Table 5.1, we could observe that the faculty decides not to use all the possible slots to build a timetable. The justification was that a whole supplementary week is being dedicated to the marking of all the examinations. We did not oppose to this decision as it sounds logical and based all our runs on the same length to allow comparisons.

On the other side, we decided to have a look at the potential impact that the addition (or removal) of workable slots can have on the objective function from the absolute minimum of slots required (Table 5.3) to the total number of slots available for the session. The values presented in Figure 5.4 below have been reached after a 4 hours run time on the FIAL-201806 dataset.

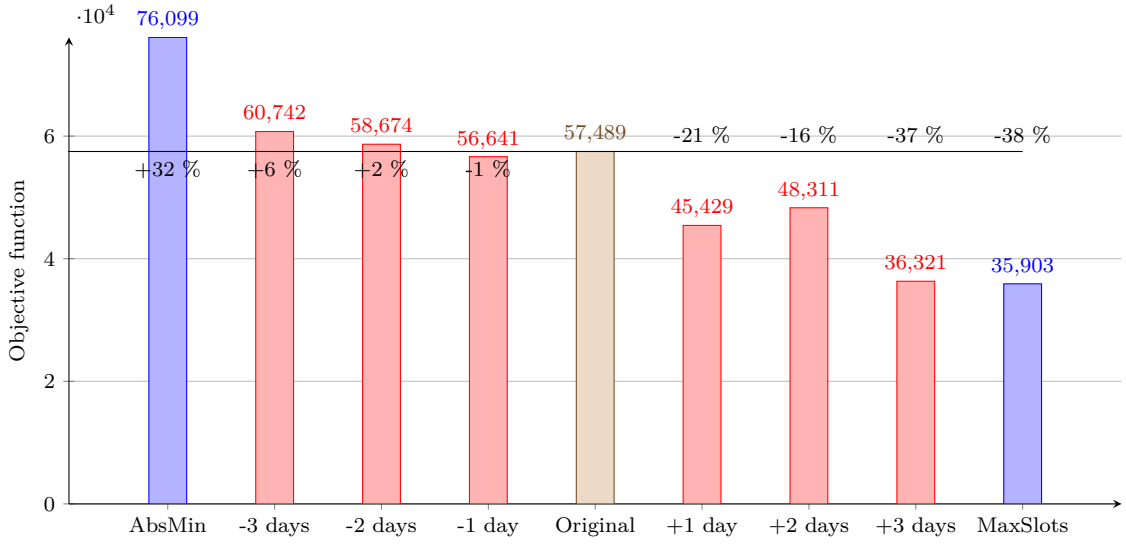


Figure 5.4.: Comparison of the objective function when adding and removing time slots.

The values observed meet globally the expectations from such alterations: the more slots you have, the better you spread, and the lower the value of the objective function will be (and vice versa). One exception can be observed though on the run with two additional days (46 slots in total) where the objective function is higher than in the default with one day added. The tests were conducted twice on the dataset and the same results was obtained twice. We could not find any plausible explanation about

this “awkward behaviour” as it did not occurred after that. Another point of interest is the where one day has been taken off and that action puts the objective function below the original case. We have a twofold theory about this subject: as the removed day was a Saturday that counted merely 10 exams in the original situation (see Figure 5.2e), the penalty involved in the transfer was quite limited and the reduction of the search space helped to visit new nodes.

Besides that, the penalty difference is more important on the right hand side (when adding new slots) than it is when removing slots. It should be noted though that in the original configuration, the last day is a Saturday meaning that the addition of two new available slots require in fact four new slots (otherwise the first two would be on Sunday), see Table 5.5 for more details about the number of slots that are available in each case. As the variation seems quite limited, making the examination more compact does not seem problematic, and could be envisaged, at least if this trend was to be confirmed in a case where all the data necessary was available.

	AbsMin	-3d	-2d	-1d	Original	+1d	+2d	+3d	···	MaxSlots
# Slots	32	34	36	38	40	44	46	48	···	54

Table 5.5.: Number of slots according to the length of the session.

Conclusion

In this thesis, we did a global overview of the modelling techniques and methodologies that can be envisaged to create examination timetables. In order to make a model that is proportionate to the requirements and consistent through all the datasets, we decided to use an Integer Programming (IP) formulation as it has already been used frequently earlier in our studies before implementing it into AIMMS, an integrated development environment focused on optimization problems.

The model we built is functional, meaning that the formulation is capable of creating a feasible examination timetable on short notice without reaching optimality though. It must be noted that the input data we used for our tests are partial and only fulfil some of the requirements (conflict-free for both students and teachers, consecutive exams and predetermined slots configuration) while others stay out of the scope (teacher availabilities, pre-constrained exams and enforced simultaneity of exams) as they could not be recovered from the individual tables.

On top of being functional and while keeping in mind the remark about the data, the quality of the solution seems to improve when using our model both on the aspect of minimizing the objective function and the increase of the Jain Fairness Index, used to measure the equity of the penalty distribution between the students.

On the other hand, several issues lie in the implantation of our model in real terms. Indeed, as the tool and solver we used are very expensive unless being utilized for academic research and as some of the data treatments are still to be performed manually, it would require both a budget and some adjustments to the present procedure to reap benefits that remain, in the current state of the operations, purely hypothetical.

Future research

During the time we spent on this thesis, we found several directions that could be analysed in future research about examination timetabling problems.

- One should attempt to create a model that could take care of all the different faculties at once as the number of inter-curricular courses tend to increase. This lead could improve the allocation of the resources given that the problem would be treated as a whole and not externalize them as constraints in individual models.
- As exams that follow each other have rarely the same level of difficulty and that students have preferences about their sequence. For instance, in a case with two examinations separated by 1 day, the student should more likely prefer one easy

exam after a hard one than a hard exam after a easy one. This would require a scale to quantify the difficulty.

- Taking care of the assignment of examination to rooms is possible as we experimented it partially on our datasets (but did not disclosed it in this thesis) but is hard to put into practice as the resolution is even more complex than the basic formulation we already have. It would require more computational power (or time) to do so as the search space would increase hundredfold.

Bibliography

Bardadym, V. A. (1996). Computer-aided school and university timetabling: the new wave. In G. Goos, J. Hartmanis, J. van Leeuwen, E. K. Burke & P. Ross (Eds.), *Practice and theory of automated timetabling* (Vol. 1153, pp. 22–45). Lecture Notes in Computer Science. doi:10.1007/3-540-61794-9_50

Beale, E. M. L. & Forrest, J. J. H. (1976). Global optimization using special ordered sets. *Mathematical Programming*, *10*(1), 52–69. doi:10.1007/bf01580653

Bergmann, L. K., Fischer, K. & Zurheide, S. (2014). A linear mixed-integer model for realistic examination timetabling problems. In E. Özcan, E. K. Burke & B. McCollum (Eds.), *Patat 2014* (pp. 82–101).

Bisschop, J. & Roelofs, M. (2004). The modeling language AIMMS. In *Applied optimization* (pp. 71–104). doi:10.1007/978-1-4613-0215-5_6

Burke, E. K., Bykov, Y., Newall, J. & Petrovic, S. (2004). A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, *36*(6), 509–528. doi:10.1080/07408170490438410

Burke, E. K., Bykov, Y. & Petrovic, S. (2001). A multicriteria approach to examination timetabling. In G. Goos, J. Hartmanis, J. van Leeuwen, E. K. Burke & W. Erben (Eds.), *Practice and theory of automated timetabling iii* (Vol. 2079, pp. 118–131). Lecture Notes in Computer Science. doi:10.1007/3-540-44629-X_8

Burke, E. K., Jackson, K., Kingston, J. H. & Weare, R. (1997). Automated university timetabling: the state of the art. *The Computer Journal*, *40*(9), 565–571. doi:10.1093/comjnl/40.9.565

Burke, E. K., Kingston, J. H. & Pepper, P. A. (1998). A standard data format for timetabling instances. In G. Goos, J. Hartmanis, J. van Leeuwen, E. K. Burke & M. W. Carter (Eds.), *Practice and theory of automated timetabling ii* (Vol. 1408, pp. 213–222). Lecture Notes in Computer Science. doi:10.1007/BFb0055891

Burke, E. K. & Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operational Research*, *140*(2), 266–280. doi:10.1016/S0377-2217(02)00069-3

Carter, M. W. (1986). Or practice: a survey of practical applications of examination timetabling algorithms. *Operations Research*, *34*(2), 193–202. doi:10.1287/opre.34.2.193

Carter, M. W., Laporte, G. & Chinneck, J. W. (1994). A general examination scheduling system. *Interfaces*, *24*(3), 109–120. doi:10.1287/inte.24.3.109

Carter, M. W., Laporte, G. & Lee, S. Y. (1996). Examination timetabling: algorithmic strategies and applications. *The Journal of the Operational Research Society*, *47*(3), 373. doi:10.2307/3010580

Cook, W. J. (2012). *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton, N.J.: Princeton University Press.

Cooper, T. B. & Kingston, J. H. (1996). The complexity of timetable construction problems. In G. Goos, J. Hartmanis, J. van Leeuwen, E. K. Burke & P. Ross (Eds.), *Practice and theory of automated timetabling* (Vol. 1153, pp. 281–295). Lecture Notes in Computer Science. doi:10.1007/3-540-61794-9_66

Cowling, P., Kendall, G. & Mohd Hussin, N. (2002). A survey and case study of practical examination timetabling problems. In E. K. Burke & P. de Causmaecker (Eds.), *Practice and theory of automated timetabling iv: 4th international conference (patat 2002)* (pp. 258–261). Lecture Notes in Computer Science. Berlin: Springer.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, *19*(2), 151–162. doi:10.1016/0377-2217(85)90167-5

Diestel, R. (2017). *Graph theory*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Dimopoulou, M. & Miliotis, P. (2001). Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, *130*(1), 202–213. doi:10.1016/S0377-2217(00)00052-7

Fonseca, G. H., Santos, H. G., Carrano, E. G. & Stidsen, T. J. (2017). Integer programming techniques for educational timetabling. *European Journal of Operational Research*, *262*(1), 28–39. doi:10.1016/j.ejor.2017.03.020

Jain, R. K., Chiu, D.-M. W. & Hawe, W. R. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared computer system: dec-tr-301. Digital Equipment Corporation.

Karp, R. M. (1972). Reductibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer computations: proc. of a symp. on the complexity of computer computations* (pp. 85–103). The IBM Research Symposia Series. New York, NY: Plenum Press.

Kingston, J. H. (2014). Timetable construction: the algorithms and complexity perspective. *Annals of Operations Research*, *218*(1), 249–259. doi:10.1007/s10479-012-1160-z

Landa Silva, J. D., Burke, E. K. & Petrovic, S. (2004). An introduction to multiobjective metaheuristics for scheduling and timetabling. In X. Gandibleux, M. Sevaux, K. Sörensen & T'Kindt Vincent (Eds.), *Metaheuristics for multiobjective optimisation* (Vol. 535, pp. 91–129). Lecture Notes in Economics and Mathematical Systems. doi:10.1007/978-3-642-17144-4_4

Laporte, G. & Desroches, S. (1984). Examination timetabling by computer. *Computers & Operations Research*, *11*(4), 351–360. doi:10.1016/0305-0548(84)90036-4

McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K. & Qu, R. (2012). A new model for automated examination timetabling. *Annals of Operations Research*, *194*(1), 291–315. doi:10.1007/s10479-011-0997-x

Migrating an existing model to gurobi optimizer. (2012). Gurobi. Retrieved from http://www.gurobi.com/pdfs/modelmigration.pdf

Mittelmann, H. D. (2018). Benchmarks of commercial and noncommercial optimization software. Retrieved from http://plato.asu.edu/talks/ismp2018.pdf

Muklason, A., Parkes, A. J., Özcan, E., McCollum, B. & McMullan, P. (2017). Fairness in examination timetabling: student preferences and extended formulations. *Applied Soft Computing*, *55*, 302–318. doi:10.1016/j.asoc.2017.01.026

Müller, T. (2016). Real-life examination timetabling. *Journal of Scheduling*, *19*(3), 257–270. doi:10.1007/s10951-014-0391-z

Oude Vrielink, R. A., Jansen, E. A., Hans, E. W. & van Hillegersberg, J. (2017). Practices in timetabling in higher education institutions: a systematic review. *Annals of Operations Research*, *28*(2), 1–16. doi:10.1007/s10479-017-2688-8

Prida Romero, B. (1982). Examination scheduling in a large engineering school: a computer-assisted participative procedure. *Interfaces*, *12*(2), 17–24. doi:10.1287/inte.12.2.17

Qu, R. & Burke, E. K. (2009). Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, *60*(9), 1273–1285. doi:10.1057/jors.2008.102

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G. & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, *12*(1), 55–89. doi:10.1007/s10951-008-0077-5

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, *13*(2), 87–127. doi:10.1023/A:1006576209967

Schaerf, A. & Di Gaspero, L. (2007). Measurability and reproducibility in university timetabling research: discussion and proposals. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, . . . H. Rudová (Eds.), *Practice and theory of automated timetabling vi* (Vol. 3867, pp. 40–49). Lecture Notes in Computer Science. doi:10.1007/978-3-540-77345-0_3

Terashima-Marin, H., Ross, P. & Valenzuela-Rendon, M. (1999). Application of the hardness theory when solving the timetabling problem with genetic algorithms. In *Proceedings of the 1999 congress on evolutionary computation* (pp. 604–611). doi:10.1109/CEC.1999.781988

Weisstein, E. W. (2002). Complexity theory. Wolfram MathWorld. Retrieved June 19, 2018, from http://mathworld.wolfram.com/ComplexityTheory.html

Wolsey, L. A. (1998, December 2). *Integer programming*. Wiley-Interscience in Discrete Mathematics and Optimization. John Wiley & Sons, Inc. Retrieved from https://www.ebook.de/de/product/3599835/laurence_a_wolsey_integer_programming.html

Wren, A. (1996). Scheduling, timetabling and rostering — a special relationship? In G. Goos, J. Hartmanis, J. van Leeuwen, E. K. Burke & P. Ross (Eds.), *Practice and theory of automated timetabling* (Vol. 1153, pp. 46–75). Lecture Notes in Computer Science. doi:10.1007/3-540-61794-9_51

Appendices

1. Graph representation of the studied FIAL datasets

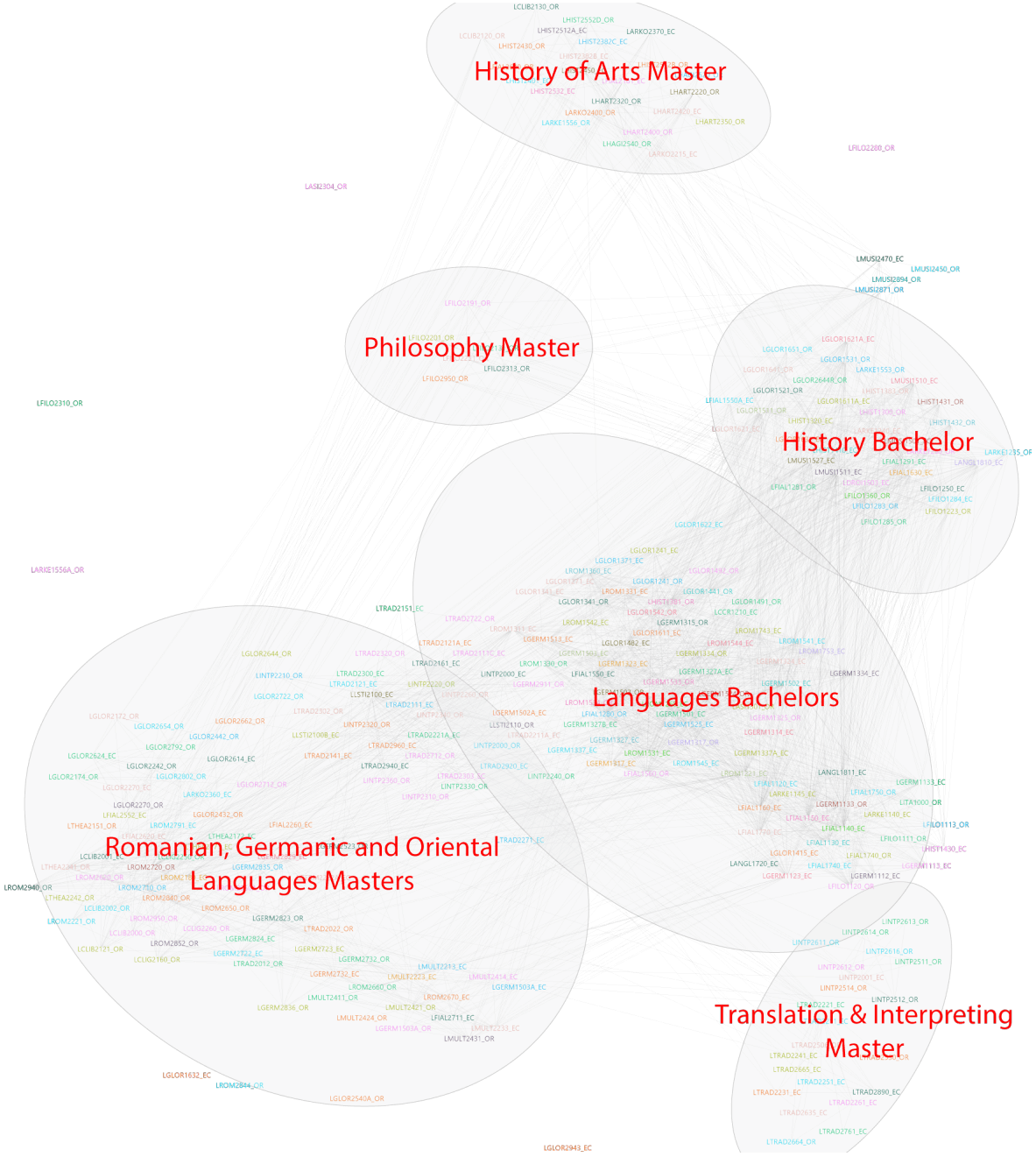


Figure 5.: Network representation of the conflicts in the **January 2017** dataset. The "clusters" provide information on the structure and grouping of the different conflicts. Six major zones are distinguishable on the graph: two relate to Bachelors (Languages and History) and four to Masters (History, Languages, Translation & Interpreting and finally Philosophy in a minor measure).

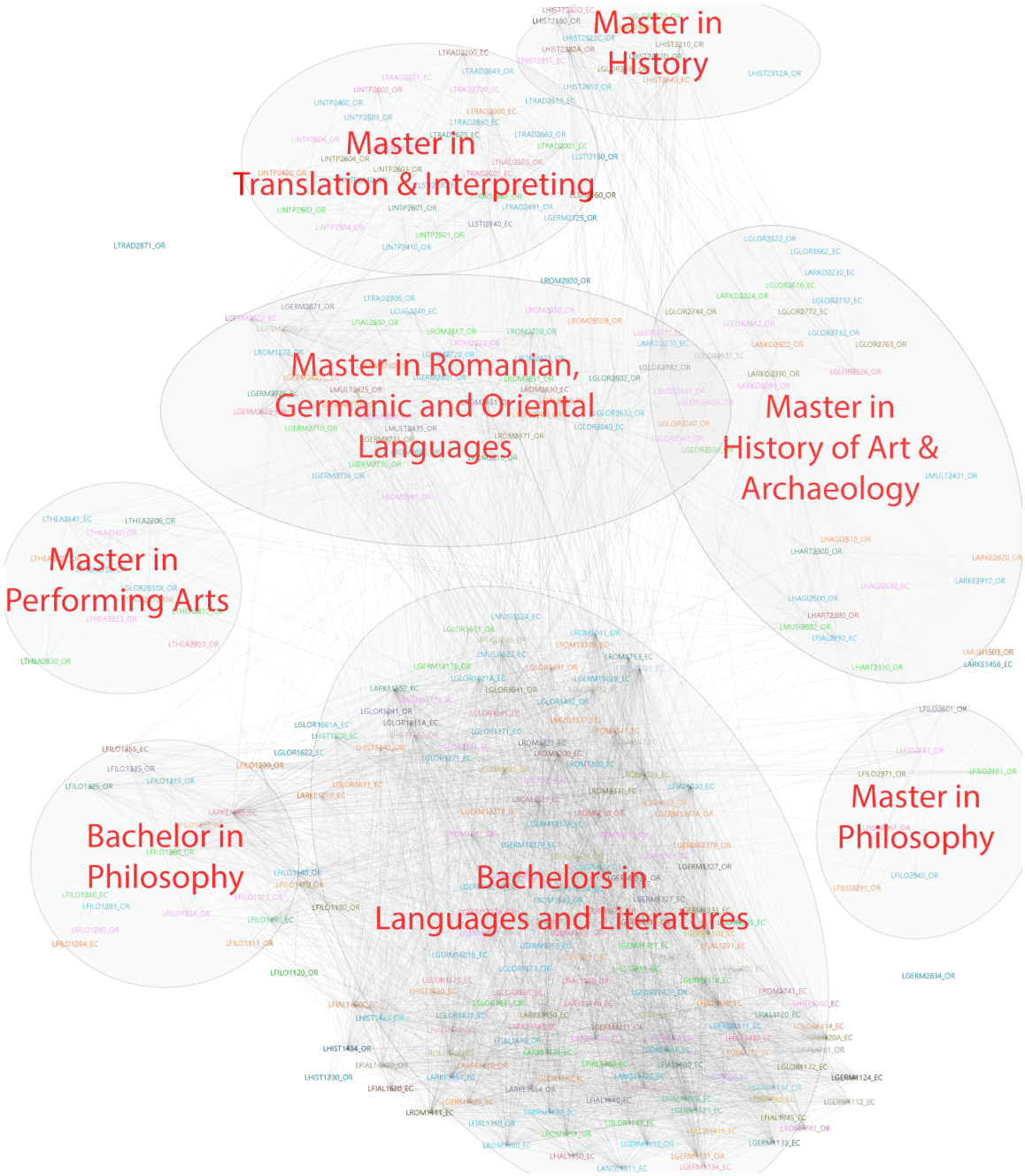


Figure 6.: Network representation of the conflicts in the **June 2017** dataset. Again, the clusters provide provide information on the structure and grouping of the different conflicts. In this diagram, the categorization is more precise for Masters than it was in January 2017. On the other hand, the "Bachelor in History" category gets absorbed into the southern part of the "Bachelors in Languages and Literatures" category.

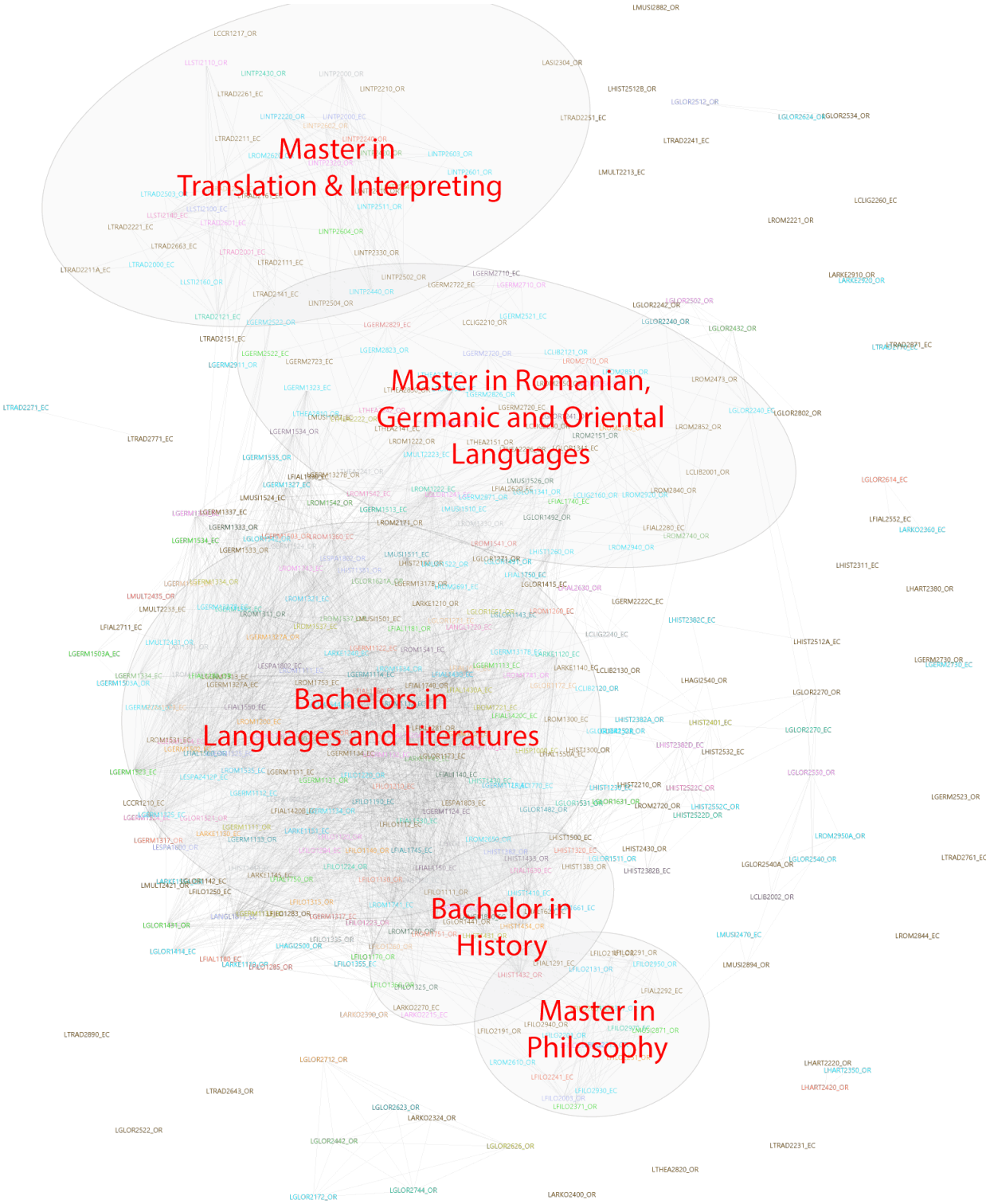


Figure 7.: Network representation of the conflicts in the **September 2017** dataset. The clustering here is more erratic and less precise than it was before: we still can observe groupings but it is now surrounded by a "galaxy" of independent examinations. The most plausible explanation is that the September's session, being only composed of resit exams, involves a lower number of examinations per student (around a quarter of the students registered have a single exam) and thus less conflicts arise.

Figure 8.: Network representation of the conflicts in the **January 2018** dataset. Even if the major categories are still to be found, the grouping is much more crumbled than it was in the previous representations.

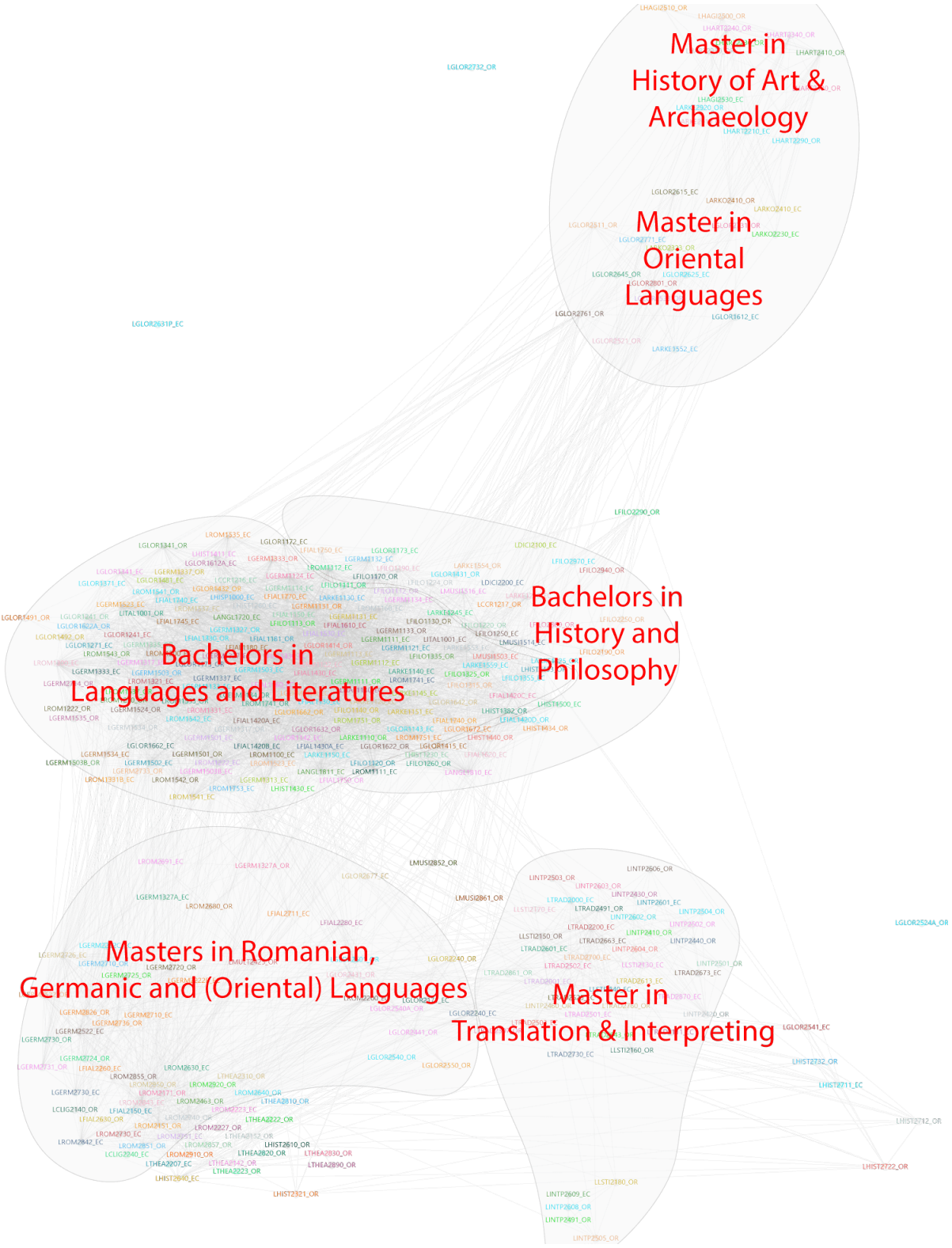


Figure 9.: Network representation of the conflicts in the **June 2018** dataset. The clusters visible on the map provide insightful information about the structure of the faculty: the "nest" at the centre represent the classes from Bachelor and the surrounding clusters (above-right, below-right and below-left) are Master students from 'History of Art & Archaeology', 'Translation & Interpreting' and 'Languages' respectively.

Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve, Belgique **www.uclouvain.be/lsm**