

# Construction of an Automated Examination Timetabling System for École Polytechnique de Louvain

Dissertation presented by  
**Adrien BROGNIET , Charles NINANE**

for obtaining the Master's degree in  
**Mathematical Engineering**

Supervisor(s)  
**François GLINEUR**

Reader(s)  
**Anthony PAPAVALIOU, Julien HENDRICKX**

Academic year 2016-2017



# ABSTRACT

---

Examination timetabling is one of the major administrative task addressed each year in educational institutions. With an increasing number of proposed courses and upcoming possibilities to combine degrees, several universities are currently using or testing automated tools designed to perform this task. Examination timetabling is a well known combinatorial optimization problem without standardized approach to solve it, but several methods proposed, as the administrative requirements strongly vary from an institution to another. This paper introduces the examination timetabling problem at École Polytechnique de Louvain-La-Neuve. We present a solver tackling the problem in multiple stages using both linear programming and meta-heuristics. This solver produces schedules respecting the administrative constraints of the faculty while maximizing the students satisfaction. It is meant to be implemented in a tool usable by the administrative support in charge of scheduling. We also explore the mathematical aspects of the problem in order to introduce techniques speeding up the resolution.



# ACKNOWLEDGMENTS

---

We are truly grateful to our Thesis supervisor, Mr. François Glineur, who provided continuous support and resources. It is highly rewarding for an engineer to apply the knowledge he acquired during his studies to tackle a real life challenging problem, so we would like to thank him for the opportunity to work on this subject.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11. We express then our profound gratitude to the CISM and CÉCI teams, without whom the considerable amount of tests would not have been possible.

More particularly, we would like to thank Mr. Damien François from the CISM team for his precious help in understanding the functioning of the CÉCI clusters.

We also thank IBM for the academic initiative allowing students to use the CPLEX<sup>®</sup> software without any charge.

Finally, we address our gratefulness to Mrs. Laurence Uwimana from the administrative support of École Polytechnique de Louvain. We appreciated the explanations she provided to help us understanding how examinations scheduling works at University.

# CONTENTS

---

|          |                                                                               |           |
|----------|-------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                                           | <b>1</b>  |
| 1.1      | Problem statement . . . . .                                                   | 1         |
| 1.2      | Thesis goals . . . . .                                                        | 2         |
| 1.3      | Thesis plan . . . . .                                                         | 3         |
| <b>2</b> | <b>Theoretical Background</b>                                                 | <b>5</b>  |
| 2.1      | Complexity theory . . . . .                                                   | 5         |
| 2.2      | University Timetabling . . . . .                                              | 7         |
| 2.3      | Graph theory . . . . .                                                        | 8         |
| 2.4      | Optimization models . . . . .                                                 | 11        |
| 2.5      | Algorithms . . . . .                                                          | 12        |
| 2.5.1    | Exact algorithms . . . . .                                                    | 13        |
| 2.5.2    | Meta-heuristics . . . . .                                                     | 15        |
| <b>3</b> | <b>Specifications</b>                                                         | <b>19</b> |
| 3.1      | Administrative specifications . . . . .                                       | 19        |
| 3.1.1    | Input - Data requirements . . . . .                                           | 19        |
| 3.1.2    | Outputs Requirements . . . . .                                                | 20        |
| 3.2      | UCT-EPL: Timetabling formulation . . . . .                                    | 21        |
| 3.3      | Key Performance Indicators . . . . .                                          | 22        |
| 3.3.1    | First approach : Gaps distribution . . . . .                                  | 23        |
| 3.3.2    | Second approach : Bad schedules rate . . . . .                                | 25        |
| 3.4      | Summary . . . . .                                                             | 27        |
| <b>4</b> | <b>Mathematical analysis of UCT-EPL</b>                                       | <b>29</b> |
| 4.1      | Parameters description . . . . .                                              | 29        |
| 4.2      | Schedule formalization . . . . .                                              | 32        |
| 4.3      | Models of UCT-EPL( $\tilde{E}$ ) . . . . .                                    | 33        |
| 4.3.1    | Linear integer program model: $M_{ILP-GD}(\tilde{E}, s(E_{basis}))$ . . . . . | 33        |

|          |                                                                             |           |
|----------|-----------------------------------------------------------------------------|-----------|
| 4.3.2    | Integer Nonlinear Program: $M_{INP-BSR}(\tilde{E}, s(E_{basis}))$ . . . . . | 37        |
| 4.4      | Problem decomposition . . . . .                                             | 38        |
| 4.4.1    | Penalty avoidable courses decomposition method . . . . .                    | 38        |
| 4.4.2    | Independent subproblems decomposition . . . . .                             | 43        |
| 4.4.3    | Nested decomposition technique . . . . .                                    | 46        |
| 4.5      | Invariants of the gaps distribution . . . . .                               | 47        |
| 4.5.1    | Minimal forced number of $Gap_{gt,gd}$ . . . . .                            | 47        |
| 4.5.2    | Minimal quantity of gaps of a subset of $S_{UGAP}$ . . . . .                | 48        |
| 4.5.3    | Clique induced gaps . . . . .                                               | 48        |
| 4.6      | Symmetry breaking . . . . .                                                 | 50        |
| 4.7      | Extra instance properties . . . . .                                         | 51        |
| 4.7.1    | Maximal general gap under non fixed courses . . . . .                       | 51        |
| 4.7.2    | Lexicographical optimization on $Gap_{gt,gd}$ . . . . .                     | 52        |
| 4.8      | Summary . . . . .                                                           | 53        |
| <b>5</b> | <b>Solver description</b> . . . . .                                         | <b>55</b> |
| 5.1      | Pre-optimization phase . . . . .                                            | 55        |
| 5.1.1    | Oral examination duplication and optimal students assignments . . . . .     | 55        |
| 5.2      | Optimization phase . . . . .                                                | 56        |
| 5.2.1    | First algorithm: CPLEX® on Integer Linear Programming model . . . . .       | 56        |
| 5.2.2    | Second algorithm: Fix And Solve . . . . .                                   | 56        |
| 5.2.3    | Third algorithm: Iterated Variable Neighborhood Search . . . . .            | 60        |
| 5.3      | Post-optimization . . . . .                                                 | 65        |
| 5.4      | Solver . . . . .                                                            | 65        |
| 5.5      | Summary . . . . .                                                           | 66        |
| <b>6</b> | <b>Tests</b> . . . . .                                                      | <b>67</b> |
| 6.1      | Data sets . . . . .                                                         | 68        |
| 6.1.1    | Methodology . . . . .                                                       | 68        |
| 6.1.2    | Description . . . . .                                                       | 69        |
| 6.2      | Parameters tuning . . . . .                                                 | 72        |
| 6.2.1    | Methodology . . . . .                                                       | 72        |
| 6.2.2    | Test configuration . . . . .                                                | 73        |
| 6.2.3    | Results . . . . .                                                           | 73        |
| 6.3      | Solvers comparison . . . . .                                                | 80        |
| 6.3.1    | Methodology . . . . .                                                       | 80        |
| 6.3.2    | Test configuration . . . . .                                                | 81        |
| 6.3.3    | Results . . . . .                                                           | 81        |
| 6.4      | Complementary tests . . . . .                                               | 82        |
| 6.4.1    | Inefficiency of CPLEX® on $M_{ILP-GD}(\tilde{E})$ . . . . .                 | 82        |
| 6.4.2    | Suppressing minor/major duplicates . . . . .                                | 83        |
| 6.4.3    | Evolution when adding or deleting time slots . . . . .                      | 84        |

---

|          |                                                  |           |
|----------|--------------------------------------------------|-----------|
| 6.4.4    | Impact of the decomposition method . . . . .     | 84        |
| 6.5      | Comparison with a real schedule . . . . .        | 85        |
| 6.5.1    | Methodology . . . . .                            | 85        |
| 6.5.2    | Results . . . . .                                | 86        |
| 6.6      | Summary . . . . .                                | 87        |
| <b>7</b> | <b>Conclusion</b>                                | <b>89</b> |
| <b>A</b> | <b>Mathematical proofs</b>                       | <b>92</b> |
| A.1      | Penalty avoidable decomposition method . . . . . | 92        |

# LIST OF FIGURES

---

|      |                                                                                             |    |
|------|---------------------------------------------------------------------------------------------|----|
| 2.1  | Example of weighted graph modeling a course timetabling . . . . .                           | 9  |
| 2.2  | Example of a graph with two connected components . . . . .                                  | 10 |
| 2.3  | Example of node coloring . . . . .                                                          | 10 |
| 2.4  | Example of Branch and Bound tree . . . . .                                                  | 14 |
| 3.1  | Examples of sessions gap distributions . . . . .                                            | 24 |
| 5.1  | Solver architecture . . . . .                                                               | 65 |
| 6.1  | Parameters tuning - 30 min - Data set 2: solvers with their best configurations . . . . .   | 74 |
| 6.2  | Parameters tuning - 30 min - Data set 9: solvers with their best configurations . . . . .   | 74 |
| 6.3  | Parameters tuning - 30 min - Data set 10: solvers with their best configurations . . . . .  | 75 |
| 6.4  | Parameters tuning - 30 min - Data set 2: pre-processing performance . .                     | 75 |
| 6.5  | Parameters tuning - 30 min - Data set 9: pre-processing performance . .                     | 76 |
| 6.6  | Parameters tuning - 30 min - Data set 10: pre-processing performance . .                    | 76 |
| 6.7  | Parameters tuning - 1 hour - Data set 2: solvers with their best configurations . . . . .   | 77 |
| 6.8  | Parameters tuning - 1 hour - Data set 9: solvers with their best configurations . . . . .   | 77 |
| 6.9  | Parameters tuning - 1 hour - Data set 10: solvers with their best configurations . . . . .  | 78 |
| 6.10 | Parameters tuning - 3 hours - Data set 2: solvers with their best configurations . . . . .  | 78 |
| 6.11 | Parameters tuning - 3 hours - Data set 9: solvers with their best configurations . . . . .  | 79 |
| 6.12 | Parameters tuning - 3 hours - Data set 10: solvers with their best configurations . . . . . | 79 |

|                                                                                                                                                     |    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 6.13 Performances profiles of the five best solvers . . . . .                                                                                       | 81 |
| 6.14 Effects of suppressing major/minor duplicates . . . . .                                                                                        | 83 |
| 6.15 Impact of adding/deleting time slots in a session - Data sets 7 and 17 . .                                                                     | 84 |
| 6.16 Comparison of the gaps distribution . . . . .                                                                                                  | 86 |
| 6.17 Comparison of the sum of gaps on subset of gaps with the minimum<br>quantity of gaps on the same subset for different subset of gaps . . . . . | 87 |

# LIST OF TABLES

---

|     |                                                                                                                             |    |
|-----|-----------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Generate&Test running time . . . . .                                                                                        | 6  |
| 2.2 | UCT categories . . . . .                                                                                                    | 7  |
| 2.3 | UCT components . . . . .                                                                                                    | 8  |
| 3.1 | Critical number of examinations per days window . . . . .                                                                   | 25 |
| 4.1 | UCT-EPL( $\tilde{E}$ ) ILP model: $M_{ILP-GD}(\tilde{E}, s(E_{basis}))$ . . . . .                                           | 36 |
| 4.2 | UCT-EPL( $\tilde{E}$ ) INP model: $M_{INP-BSR}(\tilde{E}, s(E_{basis}))$ . . . . .                                          | 37 |
| 4.3 | Linear integer programming model to identify penalty avoidable subset of courses . . . . .                                  | 42 |
| 4.4 | Constraint satisfaction problem to schedule the set of courses in $E_{pa}^k$ . . . . .                                      | 43 |
| 4.5 | Impact on the running time of the independent subproblems decomposition methods for different values of c and n. . . . .    | 46 |
| 4.6 | ILP model to compute the minimal forced quantity of $Gap_{g_t, g_d}$ . . . . .                                              | 47 |
| 4.7 | ILP model to compute the minimum quantity of gaps for a subset of $S_{UGAP}$ . . . . .                                      | 48 |
| 4.8 | Numbers of schedules before and after symmetry breaking . . . . .                                                           | 51 |
| 4.9 | Linear integer program to optimize lexicographically the number of $Gap_{g_t, g_d}$ for $(g_t, g_d) \in S_{UGAP}$ . . . . . | 52 |
| 5.1 | Solvers . . . . .                                                                                                           | 66 |
| 6.1 | Data sets 1-6 . . . . .                                                                                                     | 70 |
| 6.2 | Data sets 7-12 . . . . .                                                                                                    | 70 |
| 6.3 | Data sets 13-18 . . . . .                                                                                                   | 71 |
| 6.4 | Data sets 19-24 . . . . .                                                                                                   | 71 |

# LIST OF SYMBOLS

---

EPL : École Polytechnique de Louvain

UCL : Université Catholique de Louvain

UCT : University Course Timetabling problem

UCT-EPL : University Course Timetabling problem at École Polytechnique de Louvain

PATAT : International Series of Conferences on the Practice and Theory of Automated Timetabling

LP : Linear Program

IP : Integer Program

ILP : Integer Linear Program

INP : Integer Nonlinear Program

IVNS : Iterated Variable Neighborhood Search

FAS : Fix And Solve algorithm

CILP : CPLEX<sup>®</sup> on Integer Linear Programming model

CUT : Coordinated Universal Time

KPI : Key Performance Indicator

$P^{GD}$  : Gap Distribution penalty function

$BSR$  : Bad Student Rate

$P^{BSR}$  : Bad Student Rate penalty function

$M_{ILP-GD}$  : Gap Distribution model

$M_{INP-BSR}$  : Bad Student Rate model

UCT-EPL-GD : UCT minimizing  $P^{GD}$

UCT-EPL-BSR : UCT minimizing  $P^{BSR}$



# INTRODUCTION

---

## 1.1 Problem statement

École Polytechnique de Louvain (EPL) is the engineering faculty at Université Catholique de Louvain (UCL). Each year at EPL, a major duty of the administration is to plan the examinations schedule. This task involves more than 1700 students enrolled, 150 to 200 courses organized by the faculty and about 100 courses proposed by other faculties to EPL students. Courses are scattered over 2 semesters, at the end of which a session that lasts between 3 and 4 weeks is organized. Each day of the session is divided into 2 time slots at which examinations can be assigned, one in the morning and one in the afternoon, and examinations can be scheduled at any day of the week except on Sundays. An additional session takes place in August, so that students can represent failed examinations. At the moment, examinations schedule are generated manually. Over the years, the timetable planners have developed several strategies in order to build smart schedules giving time for students to study their courses. Nevertheless, some students might end up having two examinations at the same time, a situation further denoted as a conflict, which sometimes implicates several manual refinements in order to reach a viable schedule.

In addition to simply assigning an examination to a time slot, schedules must follow several administrative rules specific to the faculty, as described below.

First, EPL students can follow courses proposed by other faculties. These courses are scheduled independently of the decision-making power of the faculty and this information is sometimes available after the schedule has been made, leading to conflict situations. Symmetrically, students from other faculties can follow courses from EPL and the secretary must then avoid scheduling these courses at the same time as others courses followed by one of these students in his faculty. One could pertinently argue that it would be more advantageous for UCL to plan the examinations from all the faculties together. However, this is not the case currently due to multiple differences

between the administrative processes and we will make the assumption in the present Thesis that the examinations schedules of other faculties are already planned when we tackle the problem of making the schedule for EPL.

Secondly, bachelor students must choose at the beginning of their second year a minor and a major option. Many courses from minor options are given outside EPL and symmetrically some courses from EPL belong to minor options from other faculties. To avoid conflicts between major and minor courses, major examinations are organized anytime except the last week, called the minor week, while courses from minor options are actually scheduled during this week. Courses that can be taken in a major or a minor option have the particularity to be duplicated: one during the minor week and one during the other weeks. Often when conflicts cannot be avoided, a student is then proposed to take the minor examination instead of the major one or conversely.

A last particularity is that some examinations are oral. Written examinations will always be scheduled at one and only one time slot. For oral examinations we define a maximum capacity of students per time slot and we duplicate the examination a sufficient number of time such that all students can pass the course and the capacity is not exceeded for each duplicate. These duplicates can then be treated like normal written examinations with the additional condition that they must not be scheduled at the same time slot in the session.

## 1.2 Thesis goals

Nowadays, timetabling at EPL is performed manually. This represents a considerable amount of work and each year the examinations schedule must be modified several times because of conflict situations. Since the number of students and the possibilities to combine courses are increasing, this situation is unlikely to improve. Creating a tool performing this task efficiently could resolve this issue. The primary aim of the Thesis is then to find a solver that realizes the following:

Provide an **efficient schedule** for any given **examination session**

In addition to this main goal, we wish to achieve two other subsidiary targets :

1. Design a tool that implements the solver. This tool would be a computer desktop application, user-friendly and easily usable by the administrative support.
2. Study how modifying some administrative rules, such as increasing the number of days in a session, would affect the quality of a session.

For the purpose of the present Thesis, an examination session is provided as a data set structured by several elements described Section 3.1 - Inputs Requirements. This

data set contains all the necessary information, such as the courses, the inscriptions or the dates of a session. A schedule is an assignation for each examination to a specific time slot. The purpose is to design a solver that takes an examination session as input and return an efficient schedule in the sense of the definition below.

An efficient schedule is a schedule not only meeting the aforementioned administrative rules but also maximizing the satisfaction of the students with their schedule. This is done under the frame of two important considerations:

- All students must be able to pass their examinations during a session, i.e. conflicts between two examinations must be rare cases. Note that if conflicts cannot be avoided, an off session or an extra oral examination can be organized for students in this case. So avoiding conflicts is a priority, but it is possible to consider a session with one or two students with a conflict if it allows to obtain a much better session in return.
- The second prior goal is to maximize the spacing between each pairs of examinations of a student so that he has enough time to study.

We choose an approach where the solver maximizes the global satisfaction, while ensuring that no students will end up with an extremely bad schedule. We will propose a way to measure the global satisfaction based on our own student experience and recommendations granted by the administrative support at EPL.

With regards to the secondary aims, rather than computing a new general algorithm that could be applied to multiple university scheduling problems, we will focus on the creation and validation of a solver suitable for a real case of examination timetabling at EPL. The idea here is not to assess the performance of an approach compared to another, as in most of the recent papers [QBML08], but to build a tool that can be used by a university administrative support.

## 1.3 Thesis plan

First of all, we introduce the state of the art framework of university timetabling and a series of mathematical concepts useful for tackling these kind of problems. We will then present the specifications induced by the statement. These specifications formalize the different components of the problem and list the objectives. From these specifications, it will also be possible to define a series of key performance indicators, evaluating the quality of a schedule. A chapter is then dedicated to the mathematical study of the problem in order to extract results that may improve the performances of the algorithms. This is followed by the main part of the report entitled Solver Description. It describes the different algorithms used by the solver as well as a motivation on the different modeling choices and parameters tuning. The final chapter before the

conclusion reports all the tests conducted to define the most adequate solver, thanks to which it will finally be possible to describe a complete and efficient approach meant to deal with the examination timetabling problem at EPL.

# THEORETICAL BACKGROUND

---

This section defines a series of mathematical concepts that are useful when tackling timetabling problems. At the end of this chapter, any reader with a standard background in applied mathematics and statistics should be able to understand the rest of the Thesis, whether he is familiar or not with Timetabling in Operations Research. After a few notions on time complexity, it presents the state of the art definitions of University Timetabling, as given in the recent scientific literature. It also gives relevant notions in graph theory which is the main approach used to visualize and analyze the problem. We also define the main categories of optimization programs considered when modeling timetabling problems. Finally, a survey on the main algorithms given in the literature is presented for solving these models.

## 2.1 Complexity theory

Before defining what a university timetabling problem is, it is essential to illustrate why these kind of problems might be hard to solve in terms of computational time.

**DEFINITION 1.** The **Algorithmic complexity**  $C_A$  of an algorithm  $A$  is defined as the number of instructions needed to solve any instance of the problem with size  $n$ .

Landau's notation  $\mathcal{O}$  is often used to define an upper bound on the complexity. An algorithm  $A$  is said to be of complexity  $\mathcal{O}(f(n))$  if  $\exists M > 0, \exists n_0$  such that  $\forall n > n_0, C_A \leq Mf(n)$ .  $A$  is said to be **Polynomial** if  $f(n)$  is a polynomial function.

Considering that a decision problem is a problem to which the answer is either a yes or a no, it is possible to define 3 classes of algorithms :

**DEFINITION 2.** A problem is in **class P** (polynomial time) if a polynomial algorithm that solves it exists.

**DEFINITION 3.** A decision problem is in **class NP** (non-deterministic polynomial time) provided that verifying that the solution is valid can be done in a polynomial time.

DEFINITION 4. A decision problem is said to be **NP-Complete** if it belongs to class NP but not to class P, i.e. no polynomial algorithm that solves it has been found.

Timetabling problems are combinatorial problems, i.e. they consist of finding an optimal object from a finite set of objects, known to be NP-Complete [CK95]. To illustrate how important it is to design efficient algorithms for solving timetabling problems, let consider the short Example 1.

EXAMPLE 1. Let consider an examination session of  $m$  courses and  $n$  time slots, i.e. of size  $m \times n$ . If any examination can be assigned to any time slot, they are then  $n^m$  possible schedules. A logical approach to find an optimal schedule would be the Generate&Test algorithm. It simply generates all possible courses-time slots assignment and check which one is the best according to an evaluation function. If we consider generating a schedule and comparing it to the best one as a unique operation, it has a complexity  $\mathcal{O}(n^m)$ . If an operation takes a nanosecond, one can observe in Table 2.1 that even for small instances, this algorithm is not applicable and that the running time rapidly explodes as the size of the problem is increasing.

| # Courses \ # Time slots | 5                     | 10                      | 20                      |
|--------------------------|-----------------------|-------------------------|-------------------------|
| 5                        | $3 \cdot 10^{-6}$ sec | $10^{-4}$ sec           | 0.003 sec               |
| 10                       | 0.01 sec              | 10 sec                  | 3 hours                 |
| 20                       | 26 hours              | 3000 years              | $3 \cdot 10^9$ years    |
| 30                       | 30 000 years          | $3 \cdot 10^{13}$ years | $3 \cdot 10^{22}$ years |

Table 2.1: Generate&Test running time

In practice, it is then impossible to take all the possible schedules into account and select the best one. Due to the NP-completeness of the problem and the strong dependence to the institution administrative constraints, no standard algorithm exists to solve university timetabling problems. However, researchers have developed a standard framework to define these problems, as explained in the next section, and several algorithms have been designed to solve particular or standardized instances [QBML08] [Lew08] [Sch99].

## 2.2 University Timetabling

The examination scheduling task at EPL belongs to the category of University Course Timetabling (UCT) problems in the literature[BJKW97]. These problems consist of scheduling a set of lectures for each course within a given number of rooms and time slots. These assignments must comply several constraints, that strongly vary from a University to another. Every two years, The International Series of Conferences on the Practice and Theory of Automated Timetabling (PATAT) is held, serving as a forum for an international community of researchers on all aspects of computer-aided timetable generation. University timetabling has been frequently approached and competitions were organized on particular instances to assess the performances of several solving methods [SSM03] [ABM07] [RDSB<sup>+</sup>02]. Since the 2008 conference, the UCT problems are divided into three subcategories given in Table 2.2 :

**Examination Timetabling** This category include all the examinations scheduling problems. It requires to know the students inscriptions to compute quality schedules.

**Post Enrollment Course Timetabling** Similarly to the examination timetabling, a weekly courses schedule is defined according to students inscriptions.

**Curriculum-based Course Timetabling** The courses are scheduled according to curricula. A curriculum is a set of courses attended by a given group of students.

Table 2.2: UCT categories

A usual format is often used for the UCT problems [QBML08] in order to simplify and standardize the definitions of their components:

- **Events** The elements to be scheduled (typically the courses)
- **Times** The possible time slots for allocation.
- **Resources** The resources available for assignment (typically the rooms).
- **Constraints** A set of constraints linking the three previous elements. There are two types of constraints :
  1. Hard constraints : constraints that may not be violated to generate a feasible timetable.
  2. Soft constraints : constraints that may not be avoided. The quality of a timetable is usually evaluated by a penalty function that calculates the extent to which a timetable has violated its soft constraints.

Table 2.3: UCT components

The University Course Timetabling problem at École Polytechnique de Louvain, further denoted as the UCT-EPL, is an examination timetabling problem. It will be defined in the sense of Table 2.3 in the specifications chapter.

## 2.3 Graph theory

We present here graph theoretical concepts that will be used in the mathematical analysis chapter in order to visualize and analyze the problem in an appropriate way.

**DEFINITION 5.** A **Graph**  $G := (V, E)$  is defined by a set of nodes  $V$  and a set of edges  $E \subseteq V \times V$  that represent the relationships between the nodes. The graph is a **Complete graph** if and only if for all  $i \in V, j \in V$  and  $i \neq j$  then  $(i, j) \in E$ .

One can also define a direction on the edge if needed, i.e induce  $(i, j) \neq (j, i)$  for all  $(i, j) \in E$  and the graph is then a **Directed graph**.

**DEFINITION 6.** For every  $(i, j) \in E$ , one can attribute a real number  $w(i, j)$ , denoted as the **Weight** of  $(i, j)$ . The graph is then a **Weighted graph**.

The graph representation is very useful in UCT. We often represent the problem with the courses as the nodes, with an edge between two nodes if at least one student has subscribed to both courses. The weight attributed to each edge is then the number of students taking the two courses. This gives a proper graphical representation of the problem and highlights some important properties, as discussed later. For the rest of Thesis, we will denote this graph as the **Conflict Graph** of the timetabling problem. An

example of such a graph is given on Figure 2.1. For instance, MATHS and PHYSICS are two nodes linked by the edge (MATHS, PHYSICS) of weight 7, meaning that 7 students have subscribed to both courses.

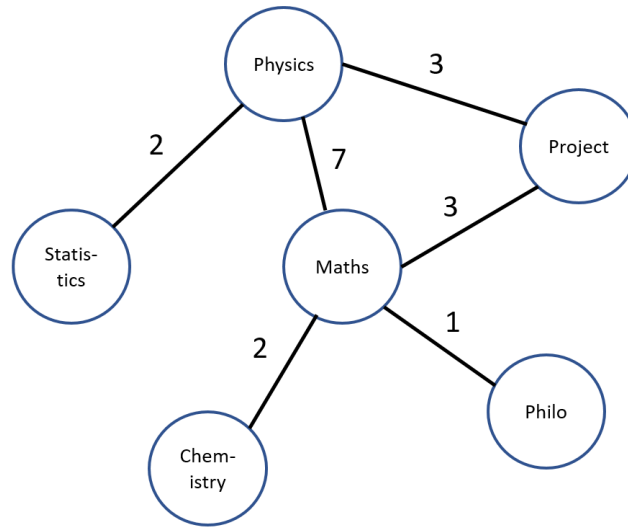


Figure 2.1: Example of weighted graph modeling a course timetabling

**DEFINITION 7.** A **Subgraph**  $\tilde{G} := (\tilde{V}, \tilde{E})$  of  $G := (V, E)$  is a graph such that  $\tilde{V} \subset V$  and  $\tilde{E}$  is the subset of  $E$  obtained by removing all the edges adjacent to  $V \setminus \tilde{V}$ .

**DEFINITION 8.** A **Path** on a graph  $G := (V, E)$  is a vector of nodes  $(v_0, v_1, \dots, v_k) \in V^k$  such that  $(v_i, v_{i+1}) \in E$  for  $i = 0, \dots, k - 1$ .

**DEFINITION 9.** A graph is said to be **Connected** if for every pair of nodes  $i, j$  in  $V$ , there exists a path between  $i$  and  $j$ . If a graph is not connected, we can define a **Connected component** of the graph as a connected subgraph  $\tilde{G}$  of  $G$  such that there exist no connections between any nodes of  $\tilde{G}$  and  $G \setminus \tilde{G}$ .

This last definition is very useful to perform a problem decomposition. Indeed, if the conflict graph is not connected, it can often be decomposed into smaller subproblems modeled by the connected components of the graph, as studied in chapter 4 (see Figure 2.2).

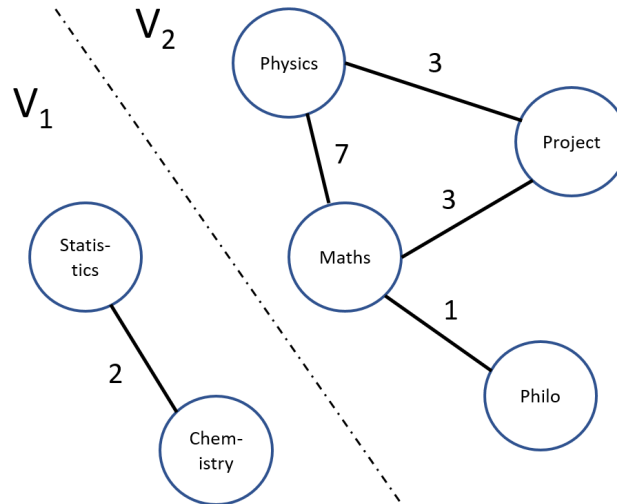


Figure 2.2: Example of a graph with two connected components

DEFINITION 10. A **Clique** in a Graph  $G := (V, E)$  is a subset  $\tilde{V}$  of  $V$  such that the subgraph induced by  $\tilde{V}$  is a complete graph.

DEFINITION 11. The **Node coloration** of a graph  $G := (V, E)$  consists in assigning to every node  $i$  of the graph a color  $c_i$  such that  $c_i \neq c_j$  for all  $(i, j) \in E$ . The minimum number of colors needed to color a graph  $G$  is called the **Chromatic Number**, denoted as  $\chi(G)$ .

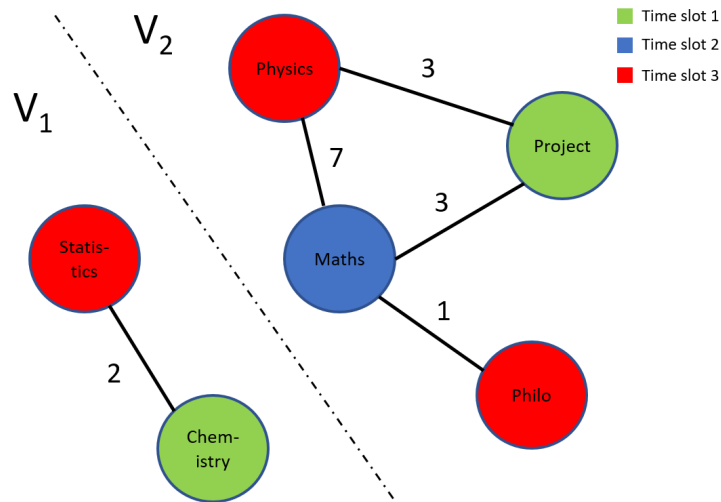


Figure 2.3: Example of node coloring

Node coloring and cliques are two very helpful notions when looking for optimal examinations schedule. By the definition of the chromatic number and a clique, we naturally observe that  $\chi(G)$  is lower bounded by the size of the largest clique. For example on Figure 2.3, the clique of maximum size is 3 and we naturally observe that the chromatic number is 3. If we consider colors as assignment to a time slot in a conflict graph, this lower bound indicates how many time slots are needed to avoid that a

student has two examinations at the same time. We will see with more details how to apply this approach in a more rigorous way to our problem.

The interest in course timetabling actually began at the end of the 1960's when Welsh and Powell found an analogy between the graph coloring problem and general timetabling problems [WP67][Woo69]. Several algorithms were developed to solve the minimum graph coloring problem and then applied on courses scheduling cases. These algorithms are easily adaptable to a large range of scheduling problems but do not offer interesting propositions when it come to model soft constraints. Nowadays, graph based algorithms have been supplanted by optimization techniques but they are still widely used for visualization and ancillary analysis.

## 2.4 Optimization models

Optimization theory is quite vast and abundant. Since we are only interested here to model timetabling problems and solve them, definitions will be limited to basic categories of models. For each of these categories, a series of specific algorithms have been designed and will be introduced in the next chapters.

**DEFINITION 12.** An **Optimization problem** is the problem of finding the minimum of a function  $f : \mathcal{X} \rightarrow \mathbb{R}$  over a set  $\mathcal{X}$ :

$$\begin{aligned} \min_x f(x) \\ \text{subject to } x \in \mathcal{X} \end{aligned}$$

$\mathcal{X}$  is the **Feasible set** of the problem and  $f$  the **Objective function**. Any  $x \in \mathcal{X}$  is a **Feasible solution**. Any  $x^* \in \mathcal{X}$  such that  $f(x^*) \leq f(x)$  for any  $x \in \mathcal{X}$  is an **Optimal solution**.

**DEFINITION 13.** The problem  $z^R = \min\{f^R(x) | x \in \mathcal{X}^R\}$  is a **Relaxation** of  $z = \min\{f(x) | x \in \mathcal{X}\}$  if :

1.  $\mathcal{X} \subseteq \mathcal{X}^R$
2.  $f^R(x) \leq f(x)$  for all  $x \in \mathcal{X}$

Optimization models can be defined more specifically using the two following definitions :

**DEFINITION 14.** A **Linear program**(LP) is an optimization problem that can be written

under the form :

$$\begin{aligned} \min_x \quad & c^T x \\ & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

with  $x \in \mathbb{R}^n$ ,  $A \in \mathbb{R}^{m \times n}$  and  $b \in \mathbb{R}^m$ . In this case, the feasible set is described by a series of  $m$  **Constraints**  $\sum_{j=1}^n a_{ij}x_j \ i = 1, \dots, m$ .

DEFINITION 15. An **Integer program**(IP), under which we distinguish Integer Linear Program(ILP) and Nonlinear Integer Program (INP), is an optimization problem with the additional constraint that  $x \in \mathbb{Z}^n$ . Conversely, the **Linear relaxation** of an IP is obtained by suppressing constraint  $x \in \mathbb{Z}^n$ . An optimization problem that is not an integer program is **Continuous**.

Since Timetabling problems are defined over discrete sets of courses, lectures and rooms, they are generally modeled as Integer programs, where the objective function gives the quality of a schedule and the feasible set accounts for the administrative constraints that may not be violated [BJKW97]. They are either represented under a linear or nonlinear form, depending on the possibility to model constraints and objectives linearly and the algorithms we want to apply to the problem.

In addition of these modeling definitions, another useful notion is introduced: the primal constraint graph.

DEFINITION 16. The **Primal constraint graph**  $G_P := (V, E)$  of an optimization problem  $P$  is a graph such that:

- each node in  $V$  coincide to a variable of  $P$
- there exists an edge between two nodes if the the corresponding variables are linked by a constraint

## 2.5 Algorithms

The second step in solving a UCT problem, after modeling it as an ILP or INP, is to develop a solving algorithm. Due to its computational complexity and the large variety of possible constraints, no standard algorithm has been proved to be the most efficient [QBML08]. Nowadays, papers are written each year to present new approaches designed to solve particular cases of this problem. Various approaches have been proposed in the last 30 years. We present here a non exhaustive list of algorithms that have

been proven to be successful recently.

Optimization algorithms can usually be classified into 3 categories:

1. **Exact algorithms** Algorithms converging towards the optimal solution.
2. **Iterative methods** Algorithms converging in a finite number of iterations towards a solution close to optimality.
3. **Heuristics or Meta-heuristics** Algorithms running during a given amount of time to find a good solution.

Iterative methods are usually used to solve non linear continuous optimization problems and have no or few applications when it comes to timetabling. The other two categories have been studied and proven to be successful so we give an overview of the most popular algorithms belonging to these categories for solving timetabling problems [QBML08].

### 2.5.1 Exact algorithms

This approach consists in modeling the problem an ILP and solving it using algorithms that converge towards optimality. It is efficient in the sense that it provides an optimal solution to the corresponding linear program. However, if the size of the problem is too large, there is a risk that the algorithm will not converge towards the optimal solution in a sufficiently short amount of time. The first integer programs modeling University timetabling problems appeared in the 1970's and 1980's. The results were encouraging when solving these IP with exact algorithms on small instances[Bre76][Tri84]. Today, a timetabling system is used at the Athens University of Economics and Business, generating good schedules by solving an IP and adjusting the generated schedule using heuristics [DM01a].

Nowadays, commercial solvers, such as CPLEX®, GUROBI® and XPRESS®, combine a series of exact algorithm to solve Integer Programs. They have been proven to be highly successful on various problems so that searchers generally prefer to use one of these off the shelf solvers rather than implementing a new exact algorithm for timetabling problems [DM01a] [BA07] [DBH04].

We will present here the exact algorithm used by the three solvers given above to solve ILP's, namely the Branch and Bound algorithm. First we need to convince ourselves that the solution of a linear relaxation problem is always lower than the original integer program.

**PROPOSITION 1.** Let  $z = \min\{c^T x | Ax = b, x \in \mathbb{Z}\}$  and  $z_{LP} = \min\{c^T x | Ax = b, x \in \mathbb{R}\}$ . Then  $z_{LP} \leq z$ .

*Demonstration.* Let  $X_1 := \{x | Ax = b, x \in \mathbb{Z}\}$  and  $X_2 := \{x | Ax = b, x \in \mathbb{R}\}$ . Since  $X_1 \subseteq X_2$ , we have :

$$z_{LP} = \min_{x_1 \in X_1} c^T x_1 \leq \min_{x_2 \in X_2} c^T x_2 = z$$

It is then possible to define the algorithm in the following way :

ALGORITHM 1. The **Branch and Bound** algorithm for an ILP  $z = \min\{c^T x | x \in X\}$ ,  $X := \{x \in \mathbb{Z}^+ | Ax = b\}$  is defined as follow :

---

**Algorithm 1** Branch & Bound

---

- 1:  $\underline{z} \leftarrow$  lower bound by solving the linear relaxation LP
  - 2:  $\bar{z} \leftarrow$  upper bound by finding a feasible point
  - 3: Update lower and upper bounds on all the current subproblems
  - 4: **if**  $\underline{z} > \bar{z}$  **then**
  - 5:     Split the feasible set  $X$  in  $k$  subsets  $X_1, \dots, X_k$ , such that  $X = X_1 \cup X_2 \dots \cup X_k$
  - 6:     Apply step 1-8 to the  $k$  subproblems  $z_k = \min\{f(x) | x \in X_k\}$
  - 7: **else**
  - 8:     Stop : Optimal solution found
- 

The Branch and Bound algorithm can be visualized as a tree scheme updated each time we compute a lower and upper bound.

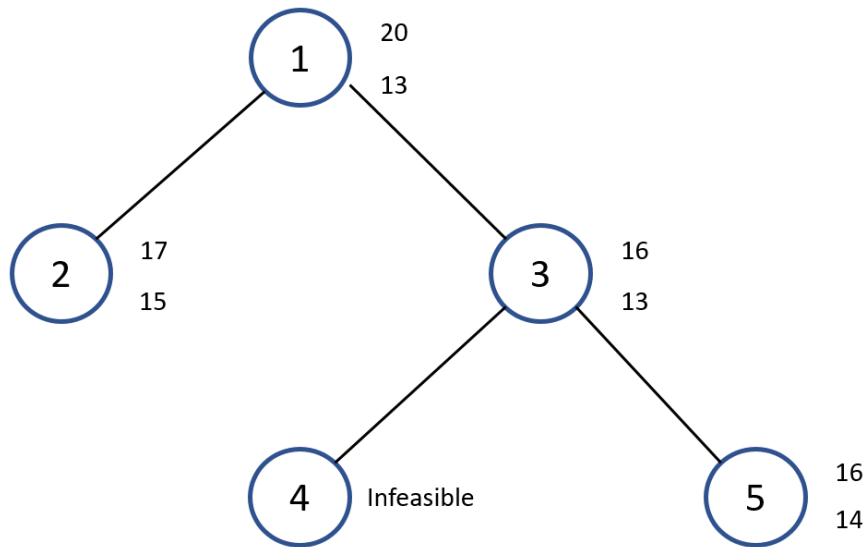


Figure 2.4: Example of Branch and Bound tree

For instance, the Branch and Bound tree depicted on Figure 2.4 is an algorithm where the initial subproblem 1 with lower bound 13 and upper bound 20 has been decomposed in a subproblem 2 and a subproblem 3 and so on. Note that the updating step (Step 3) of the lower and upper bounds is crucial. In this case the lower bound of

problem 3 becomes 14, and the upper and lower bounds of the initial problem becomes respectively 17 and 14. This update indicates which nodes of the tree are still to be explored. In this case, node 2 and 5 must be explored and bounds of the corresponding root nodes must be updated afterwards. Solving the LP relaxation is often realized using the simplex algorithm [DOW55], while finding a feasible solution can be achieved using heuristics. Note that if an initial solution, referred as a MIP start in CPLEX<sup>®</sup> with high objective is provided at the start of the algorithm, a considerable gain in time can be achieved by pruning the non relevant nodes.

Sometimes complete exploration of the Branch and Bound tree might be too long. A relevant stopping criterion depends on the gap, which is defined as follow :

DEFINITION 17. Let us consider an integer program  $z = \min\{c^T x | Ax = b, x \in \mathbb{Z}\}$ , an upper bound  $U_i \geq z$  and lower bound  $L_i \leq z$  obtained via the Branch and Bound algorithm at iteration  $i$ . The **Relative Gap**  $g_i$  at iteration  $i$  is defined as  $g_i = \frac{U-L}{U}$ .

The stopping criterion consists in choosing a tolerance  $\varepsilon$  and to stop the algorithm at iteration  $i$  if  $g_i \leq \varepsilon$ .

Solvers such as CPLEX<sup>®</sup> actually use a variant of the Branch and Bound algorithm called the Branch and Cut algorithm. They apply on the linear relaxation of the problem some iterations of the Cutting planes algorithm given below, in order to reduce the size of the feasible set without omitting the optimal integer solution.

ALGORITHM 2. The **Cutting Planes** for an IP is defined as follow :

---

**Algorithm 2** Cutting planes

---

- 1: Solve LP and get the optimal solution  $x^*$
  - 2: **if**  $x^* \notin \mathbb{Z}$  **then**
  - 3:     Add a valid inequality cutting  $x^*$  from the problem
  - 4:     Go back to 1
  - 5: **else**
  - 6:     Stop
- 

There are multiple ways to add a valid inequality cut to the relaxed problem. One of the most popular is the Gomory's Cut but solvers use many variants today [Mit02].

The branch and cut algorithm varies according to the way LP relaxations are solved, feasible points are computed and valid inequalities are added, but the main philosophy is identical and widely implemented in the most popular IP solvers.

## 2.5.2 Meta-heuristics

These are not directly applicable algorithms but more of a general framework that should be adapted regarding the problem architecture. They combine algorithms that

may provide a sufficiently good solution to an optimization problem in a short amount of time. The researches in this direction increased principally due to work of Edmund K. Burke in the 1990's [BEW94][BNW96][BJKW97]. Meta heuristics are strategies that guide the search process towards a sufficiently good solution of an optimization problem. They do not provide any guarantee on the optimality of the solution found and the only way to assess their performances is generally by comparing them with other methods, regarding chosen performances indicators.

To illustrate this definition, let us present the pseudo-code of the most basic meta-heuristic, named the Hill-Climber, applied to a minimization problem  $\min\{f(x)|x \in X\}$  :

---

**Algorithm 3** Cutting planes
 

---

```

1: function HILLCLIMBER(time : int)
2:    $x^* \leftarrow$  feasible solution
3:   while time < T do
4:      $\tilde{x} \leftarrow \min\{f(x)|x \in N(x^*) \subseteq X\}$ 
5:     if  $f(\tilde{x}) < f(x^*)$  then
6:        $x^* \leftarrow \tilde{x}$ 
7:     else
8:       Stop
9:   Return  $x^*$ 

```

---

The idea here is to set an initial solution and then, at each iteration, to define a neighborhood  $N(x^*) \subseteq X$  of feasible points depending on the current solution and select the best feasible solution from this neighborhood. In this meta-heuristic for instance, the neighborhood set can only be defined in the frame of a particular problem and does not have a general applicable definition.

The main advantage of meta-heuristics is that we do not have to use a traditional representation  $z = \min\{f(x)|x \in X\}$  to model our problem and then send it to a solver, such as CPLEX<sup>®</sup>. Their goal is to provide efficient solutions fastly, according to predefined criteria. For this reason, meta-heuristics generally provide good feasible solutions within a small amount of time and can be easily implemented. On the other hand, the optimization is often stochastic and there is absolutely no guarantee to find a global, or even local, optimal solution. In the case of the Hill climber for example, if the current  $x^*$  is better than any points  $x \in N(x^*)$ , the algorithm will stop even if  $x^*$  is not optimal.

There is no clear partition for heuristic algorithms. Nevertheless, a distinction can be made between local search techniques and the others. We will not present all the complete algorithms but rather a general functioning of the most popular meta heuristics.

### Local search techniques

Local search techniques are iterative heuristic techniques exploring new solutions based on a current best solution. Here are the main algorithms studied in the literature.

- **Local search or Hill Climber** is the basic algorithm that has been described above. All other local search techniques derive closely from this standard approach.
- **Simulated Annealing** [Egl90] follows the same process as the Hill Climber, except that it allows a move that deteriorates the solution, in terms of the objective function, according to a chosen probability. This probability to pick a bad point decreases exponentially with the number of iterations and improvements relative to the objective function. In contrary of the Hill Climber, it avoids being stuck in a locally good solution. The Simulated Annealing usually performs well and is often represented in PATAT competitions [RDSB<sup>+</sup>02] [RC95] .
- **Tabu search** [Glo90] uses a tabu list to recall some of the last points visited since a local search may lead to points that have already been reached previously by the search process. The generic Tabu search is not used anymore but several algorithms can be improved by adding the use of a Tabu List [HK03].
- **Variable neighborhood search** [MH97] is a local search technique where the definitions of the neighborhoods are multiple. A neighborhood is generally chosen at each iteration according to a probability distribution.

### Other meta-heuristics

Here we shortly introduce two other popular families of meta-heuristics,

- **Evolutionary algorithms or Genetic Algorithm** [Dav93] is an algorithm, as indicated by its name, inspired by the reproduction of species. Among a given population of solution, two are chosen to reproduce via a *crossover* function, then the result is improved using a *mutate* function. From an initial population, we hope then to obtain child solutions that is better than their parents. Sometimes a local search technique is used after the *mutate* function to further improve the child solutions. These kind of approaches are referred as **Memetic algorithms**. Genetic algorithms applied to timetabling were early presented in the 1990's and are still often used today [BEW94] [BNW96] [PB10].
- **Ant colony optimization** [DBS06] is another biologically inspired technique. At each iteration, ants build solutions and leave pheromones that are as high as the solution has a good quality. These pheromones indicate a certain probability to use the corresponding solution for building new ones. This process continues until only one solution with completely dominating pheromones remains. This metaheuristic is rarer for solving scheduling problems but has shown interesting results in other cases [Dor99] .



# SPECIFICATIONS

---

The specifications formalize the requirements in terms of data and objectives to achieve in order to compute a quality schedule. These objectives were determined based on a survey made among EPL students and an interview with Mrs Uwimana, in charge of scheduling at EPL. From these administrative specifications, it is possible to define UCT-EPL in the sense of Table 2.3 as well as Key Performance Indicators(KPI) measuring the quality of a session. KPI's are performance measures arbitrarily defined according to the goals we want to achieve.

## 3.1 Administrative specifications

### 3.1.1 Input - Data requirements

#### Time slots

An examination session is defined on multiple days. Inside theses days, given moments of the same duration are defined, and referred as time slots. Each time slot is described by :

- Time slot ID : an ID identifying the time slots.
- Time slot Index : an integer giving the position of the time slot in the session, relative to the beginning of the session. For instance, let us consider a session starting on Friday June the 6<sup>th</sup> and allowing two time slots per day, one in the morning and the other in the afternoon. The index of Friday morning is 1, the index of Saturday afternoon is 4 and the one of Monday morning is 7, even if no examinations are scheduled on Sunday.
- Day Index : an integer giving the the day number relative to the beginning of the session. Similarly to the time slot index, days without examinations scheduled have to be considered.

- Minor Week : an indicator notifying if the time slot belongs to the week when minor examinations can be scheduled.

## Courses

This component gathers together all the courses for which at least one student has enrolled. A course instance is defined by :

- Course ID : an ID identifying the course among the others.
- Examination format : an indicator specifying if the examination is written or oral. If the examination is oral, the maximum capacity that a teacher can assume for a unique time slot must be given, as an oral exam usually takes place on multiple time slots.
- Fixed slot : an indicator specifying if the examination scheduled time slot is fixed. If it is the case, the time slots ID should be given.
- Major/minor : an indicator specifying to which major or minor curriculum the examination belongs.
- Unavailabilities : a list of the time slots ID at which the teacher giving the course is not available.

## Students

It represents the students who registered for the session. A student is defined by :

- Student ID : an ID representing the student.
- Study year : an ID giving the studies and year of the student.
- Enrollments : a list of all the courses ID attended by the student.
- Major/minor : a indicator specifying in which minor/major pair the student has enrolled

### 3.1.2 Outputs Requirements

The schedule returned by the solver should fulfill the following criteria to be considered as efficient. The objectives are classified into three different sections and ranked by decreasing level of priority :

#### 1. Respect of the administrative constraints :

- Major and minor examinations are scheduled at their respective available time slots
- Fixed examinations are scheduled at their given time slots

- Oral examinations capacity is not exceeded
  - Examination cannot be scheduled on an unavailable time slot
  - Two oral duplicates cannot take place at a same time slot
2. Minimum conflicts level for students. A conflict occurs when a student has a pair of two examinations at a same time slot. The total number of conflicts should be minimal.
  3. Well balanced student schedules. As explained in Section 1.1 we want here to maximize the social satisfaction without penalizing significantly a few students. The challenge is then that every student has a minimum comfortable space without affecting the other students schedule too much. Two functions measuring the quality of a session in those terms are given later in the Key Performance Indicators section.

## 3.2 UCT-EPL: Timetabling formulation

From the administrative specifications given above it is also possible to define UCT-EPL in the frame of the definitions given in Table 2.3 . This definition is the leading point of the optimization models and graphs studies that we will compute in the next chapters.

- **Events**

The events are the examinations, represented by their ID

- **Times**

These are the time slots, also represented by their ID

- **Resources**

Optimizing the room is not part of the main objective. It will be performed in a post-processing phase so we will not consider it in the first place.

- **Constraints**

Constraints are defined

1. Hard constraints

- (a) *Examination Uniqueness*

An examination takes place at one and only one time slot

- (b) *Oral exclusion*

Two orals of the same course do not take place at a same time slot

- (c) *Fixed examinations*

A priori fixed examinations take places at their given time slot

- (d) *Minor examinations*

Minor examination take place at a time slot in the minor week

(e) *Major examinations*

Major examination take place at a time slot in the major week

(f) *Unavailabilities*

Examinations are not scheduled at an unavailable time slot

## 2. Soft constraints

- (a) *Period Spread* The quality of the schedule in terms of examinations spacing, as defined in the next section, should be minimized.

From these elements, computed in practice via the data described in the inputs requirements, it will possible to generate a schedule. Before presenting the KPI's measuring the period spread, we introduce here some very useful notations often found in the rest of the Thesis.

If  $E$  is the set of all the examinations and  $T$  the set of time slots, a schedule will be represented in the rest of the Thesis by a  $|E|$ -tuple  $s \in T^{|E|}$  (for instance  $s_e = t$  indicates that course the course  $e$  takes place at time slot  $t$  of the session). We can also generalize the concept of schedule by introducing the notion of partial schedule for a subset of courses  $\tilde{E} \subseteq E$ . That is a  $|\tilde{E}|$ -tuple  $s(\tilde{E}) \in T^{|\tilde{E}|}$ . Remark that,  $s \equiv s(E)$ . The set of all feasible partial schedules, i.e. the ones respecting all the hard constraints for the subset of courses  $\tilde{E}$ , will be denoted as  $\mathcal{S}(\tilde{E})$ .

So in other words, UCT-EPL consists to find a schedule  $s^* \in T^{|E|}$  such that :

$$s^* = \arg \min_{s \in \mathcal{S}} P(s)$$

for  $P$  a given penalty function measuring the period spread. The KPI's described in the following section provide this function and therefore complete the definition of UCT-EPL.

One can also consider looking for an efficient partial schedule for a subset  $\tilde{E} \subseteq E$  of the examinations. This case will be further referred as UCT-EPL( $\tilde{E}$ ).

$$s^*(\tilde{E}) = \arg \min_{s(\tilde{E}) \in \mathcal{S}(\tilde{E})} P(s(\tilde{E}))$$

This problem will be used later on in methods whose purpose is to solve UCT-EPL in a more efficient way. UCT-EPL( $\tilde{E}$ ) generalizes the UCT-EPL, in fact UCT-EPL( $E$ ) and UCT-EPL are equivalent.

### 3.3 Key Performance Indicators

From the specifications given in Section 3.2, we extract two series of KPI's that will allow us to evaluate the quality of a schedule, namely the period spread. This notion

is hard to define but crucial if one wants to fully assess what a good schedule is. The two series correspond to two different, but quite related, ways to consider the period spread.

Defining KPI's often link mathematical definitions and arbitrary design choices and might be subject to discussion and criticisms. However the goal here is to propose new ways to define the period spread considered as relevant to our problem, while some other definitions exist in the literature and might be more appropriate in other cases.

### 3.3.1 First approach : Gaps distribution

We first need to introduce a simple notion to measure the distance in time between two examinations: the  $Gap_{g_t, g_d}$ .

DEFINITION 18. In examinations schedule, a  $Gap_{g_t, g_d}$  is a case where one student has to take two examinations separated by  $g_t$  time slots indexes and  $g_d$  days indexes.

For instance, if we consider 2 time slots per day, a situation where a student has enrolled to two examinations, one taking place on a Wednesday afternoon and the other on the following Friday morning, is referred as a  $Gap_{4,2}$  ( $g_t$  denotes the distance in time slots and  $g_d$  the distance in days). Note that a  $Gap_{0,0}$  is actually a conflict as it denotes a situation where two examinations are scheduled at a same time slot. For brevity, we will use the  $Gap_{0,0}$  notation to stand for a conflict in the rest of this Thesis. Also note that a  $Gap_{g_t, g_d}$  is not defined for every pairs  $(g_t, g_d)$ . A  $Gap_{0,3}$  does not make sense in a session with 2 time slots per day for example.

The survey realized among EPL students revealed that the ideal minimum time between two examinations is 3 days. Therefore we consider an approach where the number of  $Gap_{g_t, g_d}$  for all possible  $(g_t, g_d) \in S_{UGAP} \subseteq \{0, \dots, 6\} \times \{0, \dots, 3\}$ , with  $S_{UGAP}$  the subset of well defined unwanted gaps, are the KPI's as there are 2 time slots per day in a session. These numbers are denoted for a schedule  $s$  as  $n_{Gap_{g_t, g_d}}(s)$ . Considering this framework, the goal of an algorithm is then to minimize these gaps.

Since we are dealing with multiple KPI's, we have to decide how to compare different schedules between them. Indeed, if one schedule is not Pareto dominating, i.e. does not have smaller gaps than all the other schedules considered, the choice of the best schedule depends on which KPI's are dominating the others. There are two ways to deal with this consideration :

1. **Lexicographic optimization** We put an order of importance on each KPI. When comparing two solutions, the one with the best KPI that strictly dominates the same KPI of the other solution is winning.

2. **Weighted objective** In a case with  $n$  KPI's denoted as  $KPI_i$ , a weight  $w_i$  is attributed to each  $KPI_i$  and we compute an objective function  $f = \sum_{i=1}^n w_i KPI_i$ . This function can then be used to compare solutions.

We have chosen the second approach. Even if the  $Gap_{g_t, g_d}$  for small lexicographical pairs  $(g_t, g_d)$  are more penalizing than the ones for large lexicographical pairs of  $(g_t, g_d)$ , we cannot say that one  $Gap_{1,0}$  is always worse than one hundred  $Gap_{2,1}$  for instance. Therefore we have chosen to attribute weights to each gap and to consider the weighted sum as a penalty function measuring the quality of a schedule according to these KPI's.  
:

$$P^{GD}(s) = \sum_{(g_t, g_d) \in S_{UGAP}} n_{Gap_{g_t, g_d}}(s) pen_{g_t, g_d}$$

with  $pen_{g_t, g_d}$  as chosen penalty.

The weights were determined such that a schedule is exponentially penalized by its number of  $Gap_{g_t, g_d}$ , i.e. if  $(g_t, g_d)$  are decreasing lexicographically then the penalty grows exponentially. Therefore we arbitrarily defined  $pen_{g_t, g_d}$  as :

$$pen_{g_t, g_d} = 2^{(6-g_t)(3-g_d)}$$

Figure 3.1 below shows that the penalty function with these arbitrary weights determine the good schedules, i.e the ones with  $n_{Gap_{g_t, g_d}}(s)$  decreasing exponentially when  $(g_t, g_d)$  decreases lexicographically.

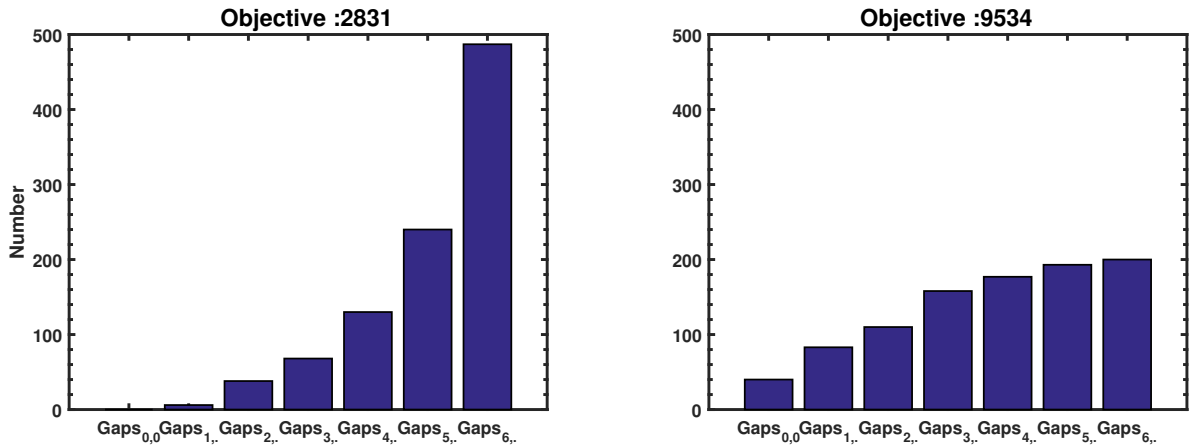


Figure 3.1: Examples of sessions gap distributions

Now that we have an adequate penalty function, we can complete the definition of UCT-EPL( $\tilde{E}$ ). This problem, defined in the sense of the penalty function  $P^{GD}$ , is denoted as UCT-EPL-GD( $\tilde{E}$ ):

$$s^* = \arg \min_{s(\tilde{E}) \in \mathcal{S}(\tilde{E})} P^{GD}(s)(\tilde{E})$$

By looking to the optimal schedule of UCT-EPL-GD( $E$ ), one could expect to find a schedule where the distances between conflicting examinations are large. However, it is possible that the unwanted gaps focus on a particular subset of the students. It is therefore possible to obtain a schedule where the KPI's on the global session are good despite the fact that a small group of students will end up with 4 examinations in 5 days for instance. So if a schedule have large  $Gap_{g_t, g_d}$  for small lexicographical pairs of  $(g_t, g_d)$ , we can declare that a session is not efficient. On the other hand, small KPI's may not absolutely guarantee that a session is good. This does not match our initial philosophy, which was to generate schedules with the best spacings between examinations without penalizing a particular subset of the students. For this reason, we decided to build another KPI measuring the percentage of "bad schedules", as it will be defined in the next section.

### 3.3.2 Second approach : Bad schedules rate

As explained before, even if the gaps numbers are low for small lexicographical pair  $(g_t, g_d)$ , some students might end up with a large number of examinations in a short amount of time, which can be considered as a bad individual schedule.

Another approach to tackle the question of the period spread is then to define a series of rules that define a bad individual period spread when violated and to check the percentage of students actually ending up with this bad schedule configuration. A bad period spread is here defined by a series of rules established in the following way : for any given time slots window we reported the minimum number of examinations that will make a schedule not satisfactory. These limits are depicted on Table 3.1. For instance, if a student has 3 or more examinations in 3 days, his schedule can be considered as bad. We consider here students with a maximum of 8 examinations (as 9 examinations or more are extremely rare cases).

| Days window    | 1 | 2-4 | 5-6 | 7-8 | 9-11 | 12-13 | more than 14 |
|----------------|---|-----|-----|-----|------|-------|--------------|
| # examinations | 2 | 3   | 4   | 5   | 6    | 7     | 8            |

Table 3.1: Critical number of examinations per days window

We can then define the Bad Schedule Rate as another KPI, which indicates if a schedule  $s$  is bad :

$$BSR(s) = \frac{\sum_{i=1}^K I_i - K^f}{K - K^f}$$

with  $K$  being the number of students,  $K^f$  the number of students with fixed bad schedule

and

$$I_i : \begin{cases} 1 & \text{if student } i \text{ as a bad schedule} \\ 0 & \text{else} \end{cases}$$

By looking for the schedule that minimizes this KPI, one could expect to obtain a perfect schedule in the sense of the rules defined above if the value of BSR is zero. On the other hand, if at least one student has an unacceptable session, it is hard to quantify how bad is the schedule. For example a situation where a student has 2 examinations in one day is not the same as having 4 examinations planed at the same time slot, even if two are considered as a bad configuration. Therefore there is also a need to build a penalty function matching with this unique KPI and reporting if an individual session is completely calamitous.

We first define two matrices representing the number of examinations at each time slot and for each student for a given schedule  $s$ .

$N_{it}(s)$  = Number of examinations taking place time slot  $t$  for which student  $i$  has enrolled

$N_{it}^f(s)$  = Number of fixed examinations taking place time slot  $t$  for which student  $i$  has enrolled

Using these two parameters, we decided to define the quality of a session using four matrices :

$$\begin{aligned} A_{it}(s) &= \prod_{k=0}^6 (\alpha^{6-k})^{N_{it+k}(s)} & A_{it}^f(s) &= \prod_{k=0}^6 (\alpha^{6-k})^{N_{it+k}^f(s)} \\ A_{it}^{Lin}(s) &= \sum_{k=0}^6 (\alpha^{6-k})^{N_{it+k}(s)} & A_{it}^{f,Lin}(s) &= \sum_{k=0}^6 (\alpha^{6-k})^{N_{it+k}^f(s)} \end{aligned}$$

The first matrices represent the workload of a student  $i$  at a given time slot  $t$ . Exponential weights are attributed at each time slot depending on how close this time slot is from an examination, using a 6 time slots horizon as more than 3 days between two examination is considered as satisfactory.

It is then possible to define a penalty function as

$$P^{BSR}(s) = \sum_{i \in S} \sum_{t \in T} [(A_{it}(s) - A_{it}^{Lin}(s)) - (A_{it}^f(s) - A_{it}^{f,Lin}(s))]$$

One can easily observe that even if only one student cumulates several examinations in a small time frame, the function will exponentially explodes. So by using this function as objective, one can expect to avoid that the inevitable bad schedules will be disastrous. On the other hand, it will also enlarge the spacing between examinations, similarly to the  $P^{GD}$  function.

Similarly than in the first approach, we can define a second variant of UCT-EPL. UCT-EPL( $\tilde{E}$ ) defined in the sense of the penalty function  $P^{BSR}$  is expressed as UCT-

EPL-BSR( $\tilde{E}$ ) :

$$s^*(\tilde{E}) = \arg \min_{s(\tilde{E}) \in \mathcal{S}(\tilde{E})} BSR(s)$$

To convince ourselves that this function matches the BSR behavior, we conducted a small experiment with  $\alpha = 1.4$  depicted on Example 3.3.2.

EXAMPLE 2. We generated 50 schedules randomly with varying quality (from the 24 data sets described in Section 6.1). We decided to rank them according to  $BSR$  and  $P^{BSR}$  and to check the absolute and maximum distances between rankings. For the two rankings  $r^{BSR}$  and  $r^{P^{BSR}}$ , the absolute distance and the maximum distances are respectively defined as :

$$d^1 = \sum_{i=1}^{50} |r_i^{BSR} - r_i^{P^{BSR}}|$$

$$d^\infty = \max_i |r_i^{BSR} - r_i^{P^{BSR}}|$$

We observed that  $d^1 = 12$ ,  $d^\infty = 3$ . It shows that this penalty function matches well the  $BSR$  and can be used to approximate this KPI.

So in addition to nicely approximate the BSR, this function appears as a good quantifier for our initial goal, which is to provide a good session for all students in terms of distance between examinations, without ending up with a few students extremely penalized for the social welfare.

## 3.4 Summary

The present chapter formalized UCT-EPL and its specifications. It shows two different ways to work on the question of the period spread. The first approach consider the gaps distribution as the measure of quality and the second one the rate of bad students. From these approaches we can extract corresponding KPI, the gaps numbers  $n_{Gap_{gt,gd}}(s)$  in the first case and the  $BSR$  in the other, as well as two penalty functions quantitatively measuring the quality of a schedule. The KPI's serve as an easy-to-visualize assessment of the quality of a session while the corresponding penalty functions serve as metrics measuring this quality. For this reason, we use the penalty functions for parameters tuning and performance assessments, while the KPI's will emphasize well the efficiency compared to a real schedule made by hand.

As shown in Section 3.3.2, the second approach is the one which better responds to our philosophy and we therefore consider  $P^{BSR}$  as the official metric for the period spread and UCT-EPL-BSR as the problem we want to solve. However, the first approach

is still relevant because, as it will be shown in the next chapter, an efficient linear formulation can easily be found.

# MATHEMATICAL ANALYSIS OF UCT-EPL

---

An essential step when working on a combinatorial problem such as UCT-EPL is to study its fundamental mathematical properties. It gives a better overview, allows to speed up its resolution and gives bounds on what we can hope to achieve in terms of the KPI's defined. The first part of this chapter is dedicated to the modeling of UCT-EPL. We will describe all the parameters related to it and formalize the notion of partial schedule and its characteristics. We will also define how to divide a complete schedule into partial schedules and how to recover this complete schedule from its decomposition. We will finally give the two different optimization models considered for solving the two variants of UCT-EPL. In the second part, we will show how these models, their associated primal constraint graph and conflict graph are useful tools to perform the following improvements :

1. Derive decomposition methods allowing to apply the resolution on partial schedules rather than the complete one.
2. Elaborate symmetry breaking constraint to solve the problem in a more efficient way.
3. Compute invariants of the gap distribution and illustrate the performance limits.
4. Introduce interesting properties of a particular instance of UCT-EPL.

## 4.1 Parameters description

This section gives the parameters attached to UCT-EPL and their notation for the rest of the Thesis. These parameters belong to 4 categories. First we specify the data set input parameters, mathematically representing all the requirements specified in Section 3.1. After that we present the design parameters arbitrarily adjustable. Thirdly, we give the notations of some useful parameters meant to model our problem in a more

convenient way. Finally we show how to compute parameters relative to a subset  $\tilde{E}$  from the original set of courses  $E$ .

### 1. Data set input parameters

- Examinations set  $E := \{1, \dots, n_E\}$
- Major examinations set  $E_{Maj} \subseteq E$
- Minor examinations set  $E_{Min} \subseteq E$
- Fixed examinations set  $E_{Fix} \subseteq E$
- Time slots set  $T := \{1, \dots, n_T\}$
- Minor week time slots set  $T_{Min} \subseteq T$
- Students set  $V := \{1, \dots, n_V\}$
- A time slots index vector  $I \in \mathbb{N}^{|T|}$  :

$$I_t = \text{index of time slot } t \quad \forall t \in T$$

- A time slots day vector  $D \in \mathbb{N}^{|T|}$  :

$$D_t = \text{day of time slot } t \quad \forall i \in T$$

- An enrollment matrix  $En \in \{0, 1\}^{|V| \times |E|}$  defined as :

$$En_{v,e} = \begin{cases} 1 & \text{if student } v \text{ enrolled for examination } e \\ 0 & \text{else} \end{cases} \quad \forall (e, t) \in V \times E$$

- Unavailability matrix  $U \in \{0, 1\}^{|E| \times |T|}$  defined as :

$$U_{e,t} = \begin{cases} 1 & \text{if examination } e \text{ may not be scheduled at time slot } t \\ 0 & \text{else} \end{cases} \quad \forall (e, t) \in E \times T$$

- Fixation matrix  $F \in \{0, 1\}^{|E| \times |T|}$  defined as :

$$F_{e,t} = \begin{cases} 1 & \text{if examination } e \text{ is fixed at time slot } t \\ 0 & \text{else} \end{cases} \quad \forall (e, t) \in E \times T$$

- Exclusion matrix  $Excl \in \{0, 1\}^{|E| \times |E|}$  defined as :

$$Excl_{e_1, e_2} = \begin{cases} 1 & \text{if examination } e_1 \text{ and } e_2 \text{ may not be scheduled at the same time slot} \\ 0 & \text{else} \end{cases}$$

$$\forall (e_1, e_2) \in E \times E$$

### 2. Design parameters.

- $max_{g_t}$  is the maximum separation of time slot indexes for conflict pairs of examinations that we want to avoid as introduced in Section 3.3.1.
- $max_{g_d}$  is the maximum separation of day indexes for conflict pairs of examinations that we want to avoid as introduced in Section 3.3.1.

### 3. Useful parameters.

- Conflict matrix  $C \in \mathbb{N}^{|E| \times |E|}$  defined as :

$$C_{e_1, e_2} = \text{number of students taking examinations } e_1 \text{ and } e_2 \quad \forall (e_1, e_2) \in E \times E$$

- An allowed placement matrix  $AP \in \{0, 1\}^{|E| \times |T|}$  defined as:

$$AP_{et} = \begin{cases} 0 & \text{if at least one of the following conditions is met:} \\ & \begin{aligned} & \bullet U_{et} = 1 \\ & \bullet e \in E_{maj} \wedge t \in T_{min} \\ & \bullet e \in E_{min} \wedge t \in T \setminus T_{min} \\ & \bullet e \in E_{fix} \wedge F_{et} = 0 \end{aligned} \\ 1 & \text{else} \end{cases} \quad \forall (e, t) \in E \times T$$

- Conflict examination pairs set  $S_C := \{(e_1, e_2) \in E \times E \mid e_1 < e_2 \wedge C_{e_1, e_2} > 0\} \subseteq E \times E$
- Unwanted Gaps set:  $S_{UGAP} := \{(g_t, g_d) \in \{0, \dots, max_{g_t}\} \times \{0, \dots, max_{g_d}\} \mid \exists (t_1, t_2) \in T \times T \text{ st. } |I_{t_1} - I_{t_2}| = g_t \wedge |D_{t_1} - D_{t_2}| = g_d\}$
- Unwanted Gaps lexicographical interval subset:  $S_{UGAP}^{a-b} := \{(g_t, g_d) \in S_{UGAP} \mid a, b \in S_{UGAP} \wedge a \leq_{lex} (g_t, g_d) \leq_{lex} b\}$  with  $a \leq_{lex} b$ . For brevity,  $S_{UGAP}^c$  is equivalent to  $S_{UGAP}^{(0,0)-c}$ .
- Conflict pairs generating unwanted gaps set  $S_{UC} := \{(e_1, e_2, t_1, t_2) \in S_C \times T \times T \mid (|I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|) \in S_{UGAP}\}$
- Excluded pairs of courses set:  $S_{EX} := \{(e_1, e_2) \in E \times E \mid e_1 < e_2 \wedge Excl_{e_1, e_2} = 1\}$

### 4. Parameters for subset of examinations $\tilde{E}$ :

- Major examinations set  $E_{Maj}(\tilde{E}) = E_{Maj} \cap \tilde{E} \subseteq E$
- Minor examinations set  $E_{Min}(\tilde{E}) = E_{Min} \cap \tilde{E} \subseteq E$
- Fixed examinations set  $E_{Fix}(\tilde{E}) = E_{Fix} \cap \tilde{E} \subseteq E$
- Conflict examination pairs set  $S_C(\tilde{E}) := \{(e_1, e_2) \in \tilde{E} \times \tilde{E} \mid e_1 < e_2 \wedge C_{e_1, e_2} > 0\}$
- Conflict pairs generating unwanted gaps set  $S_{UC}(\tilde{E}) := \{(e_1, e_2, t_1, t_2) \in S_C(\tilde{E}) \times T \times T \mid (|I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|) \in S_{UGAP}\}$
- Excluded pairs of courses set:  $S_{EX}(\tilde{E}) := \{(e_1, e_2) \in \tilde{E} \times \tilde{E} \mid e_1 < e_2 \wedge Excl_{e_1, e_2} = 1\}$

## 4.2 Schedule formalization

After specifying the notations of the parameters, we can formalize the notion of partial schedule for the subset of courses  $\tilde{E} \subseteq E$  and how to mathematically define the parameters used in the penalty functions. We remind here that the concept of partial schedule generalizes the one of schedule for the set of all examinations  $E$ . All the theory that is presented for a partial schedule also applies to a complete one. A partial schedule of UCT-EPL( $\tilde{E}$ ) is defined as a tuple  $s(\tilde{E}) \in T^{|\tilde{E}|}$ :

$$s_e(\tilde{E}) = \text{time slot } t \in T \text{ at which examination } e \text{ is scheduled} \quad \forall e \in \tilde{E}$$

A feasible partial schedule  $s(\tilde{E})$  is a schedule where the courses in  $\tilde{E}$  verifies the hard constraint given in Section 3.2. This is a schedule  $s(\tilde{E})$  s.t:

- Oral exclusion:

$$\forall (e_1, e_2) \in S_{EX}(\tilde{E}) : s_{e_1}(\tilde{E}) \neq s_{e_2}(\tilde{E})$$

- Fixed examinations:

$$\forall e \in E_{fix}(\tilde{E}) \forall t \in T \text{ s.t } F_{e,t} = 1 : s_e(\tilde{E}) = F_{e,t}$$

- Minor examinations:

$$\forall e \in E_{min}(\tilde{E}) : s_e(\tilde{E}) \in T_{min}$$

- Major examinations:

$$\forall e \in E_{maj}(\tilde{E}) : s_e(\tilde{E}) \in T \setminus T_{min}$$

- Unavailabilities:

$$\text{If } U_{e,t} = 1 \text{ for } (e, t) \in \tilde{E} \times T \text{ then: } s_e(\tilde{E}) \neq t$$

$\mathcal{S}(\tilde{E})$  is the set of feasible partial schedules for the subset  $\tilde{E}$  of examinations.

We now define two operations on partial schedules further used by the decomposition methods. We can merge two partial schedules for distinct set of courses  $\tilde{E}_1, \tilde{E}_2 \subset E$  such that  $\tilde{E}_1 \cap \tilde{E}_2 = \emptyset$ . Let  $s(\tilde{E}_1), s(\tilde{E}_2)$  be respectively the partial schedules for  $\tilde{E}_1$  and  $\tilde{E}_2$  and  $\tilde{E}_m = \tilde{E}_1 \cup \tilde{E}_2$ . If we merge these partial schedules we obtain another schedule  $s(\tilde{E}_m)$  for  $\tilde{E}_m$  with:

$$s_e(\tilde{E}_m) = \begin{cases} s_e(\tilde{E}_1) & \text{if } e \in \tilde{E}_1 \\ s_e(\tilde{E}_2) & \text{if } e \in \tilde{E}_2 \end{cases} \quad \forall e \in \tilde{E}_m$$

We can also extract a partial schedule  $s(E_{ex})$  from another partial schedule  $s(E_{init})$  with

$$E_{ex} \subset E_{init}.$$

$$s_e(E_{ex}) = s_e(E_{init}) \quad \forall e \in E_{ex}$$

We now formalize characteristics earlier introduced in Section 3.3 used in the penalty functions to measure the quality of a partial schedule.

- $n_{Gap_{g_t, g_d}}(s(\tilde{E}))$ , the number of  $Gap_{g_t, g_d}$  of a partial schedule  $s(\tilde{E})$ :

$$n_{Gap_{g_t, g_d}}(s(\tilde{E})) = \sum_{(e_1, e_2) \in Q(s(\tilde{E}), g_t, g_d)} C_{e_1, e_2}$$

where  $Q(s(\tilde{E}), g_t, g_d) = \{(e_1, e_2) \in S_C(\tilde{E}) \wedge |I_{s_{e_1}(\tilde{E})} - I_{s_{e_2}(\tilde{E})}| = g_t \wedge |D_{s_{e_1}(\tilde{E})} - D_{s_{e_2}(\tilde{E})}| = g_d\}$ .

- The number of examinations taking place at time slot  $t$  for which student  $v$  has enrolled:  $N_{v,t}(s(\tilde{E}))$ .

$$N_{v,t}(s(\tilde{E})) = |\{e \in \tilde{E} | En_{v,e} = 1 \wedge s_e(\tilde{E}) = t\}|$$

- The number of fixed examinations taking place at time slot  $t$  for which student  $v$  has enrolled:  $N_{v,t}^f(s(\tilde{E}))$ .

$$N_{v,t}^f(s(\tilde{E})) = |\{e \in E_{fix}(\tilde{E}) | En_{v,e} = 1 \wedge s_e(\tilde{E}) = t\}|$$

### 4.3 Models of UCT-EPL( $\tilde{E}$ )

In this section we will present two different optimization models whose purpose is to generate an efficient partial schedule  $s(\tilde{E})$  of UCT-EPL( $\tilde{E}$ ). We recall that UCT-EPL-BSR( $\tilde{E}$ ) has been chosen to be the main problem to solve. However we also present a model for UCT-EPL-GD( $\tilde{E}$ ) because a solver combining the two approaches is the most efficient, as we will show in the test chapter.

We will first present an ILP modeling UCT-EPL-GD( $\tilde{E}$ ), built because there exist efficient solvers for ILP models. In counterpart we loose a degree of freedom in the modeling of constraints and objective function to respect linearity. After that we introduce a second INP modeling UCT-EPL-BSR( $\tilde{E}$ ), that has the advantage to allow freedom in the modeling but whose resolution is much more complicated.

#### 4.3.1 Linear integer program model: $M_{ILP-GD}(\tilde{E}, s(E_{basis}))$

The first model we give is a linear binary program using  $P^{GD}$  as an objective function. This choice is considered because it would be very fastidious to build a linear integer program meant to find a partial schedule that minimizes  $P^{BSR}$ . In fact, it would imply

using a huge number of binary dummy variables to make the penalty function  $P^{BSR}$  linear. We sketch here the number of variables needed in the worst case to show that such a model would be intractable:

- # Binary variables to model the schedule:  $|\tilde{E}| \times |T|$
- # Binary variables to count the number of examinations for each student at each time slot:  $|V| \times |T| \times n_{exam-stud}^{max}$  where  $n_{exam-stud}^{max}$  is the number of examinations of the student that has the most enrollments for this session.
- # Binary variable for each time slot and each student to notice in which configuration are the examinations in the 6 time slots after the considered time slot:  $|V| \times |T| \times (n_{exam-stud}^{max})^6$

For a classical UCT-EPL instance, with 2000 students, 30 time slot and  $n_{exam-stud}^{max} = 8$  we would have  $1,57 \times 10^{10}$  binary variables.

Instead, we will focus here to solve UCT-EPL-GD( $\tilde{E}$ ) because the penalty function  $P^{GD}$  can be modeled with a linear function depending on a much more reasonable number of binary variables. The number of variables we need can be sketched again :

- # Binary variables to model the schedule:  $|\tilde{E}| \times |T|$
- # Binary variables to notice if two conflict courses are scheduled generating an unwanted gap:  $|S_C(\tilde{E})| \times |S_{UGAP}| \leq \frac{|\tilde{E}|^2}{2} \times (max_{g_t} + 1) \times (max_{g_d} + 1)$ .

For large instances of UCT-EPL, with 300 examinations, 30 time slots, 2000 pairs of courses in conflict and 15 unwanted gaps, we would have a total number of 39000 binary variables which is much more tractable. One could argue on the utility of this model since we explained in the previous chapter that an optimal partial schedule of UCT-EPL-GD( $\tilde{E}$ ) could lead to unmanageable examination sessions for a minority of students. Anyway we can consider this approach because finding a good gaps distribution also intuitively provides a good schedule in the sense of  $P^{BSR}$  for smaller time windows. Therefore we can hope that for a sufficient time duration, the use of efficient off the shelf solvers applied to this model can lead to schedules that outperform nonlinear models minimizing  $P^{BSR}$ . It is actually often the case for classical instances of UCT-EPL as we will see empirically later in Section 6.2 of the tests chapter.

The first step for modeling is the choice of how to represent a partial schedule with integer decision variables. To stay in line with our previous definition , we could model a partial schedule with  $|\tilde{E}|$  integer variables, each with a domain of size  $|T|$ . However, for the ease of modeling linear constraints and objective functions in an ILP, it is always easier to model decision variables with binary integer variables. We have then:  $s(\tilde{E}) \in \{0, 1\}^{|\tilde{E}| \times |T|}$ :

$$s_{e,t}(\tilde{E}) = \begin{cases} 1 & \text{if exam } e \text{ is assigned to time slot } t \\ 0 & \text{else} \end{cases} \quad \forall (e, t) \in \tilde{E} \times T$$

Additionally, since we search only for feasible partial schedules that respect the hard constraints of UCT-EPL( $\tilde{E}$ ), we rewrite the formalization of a feasible partial schedule with the introduced binary representation:

- Examination uniqueness:

$$\sum_{t \in T} s_{e,t}(\tilde{E}) = 1 \quad \forall e \in \tilde{E}$$

- Oral exclusion:

$$\forall (e_1, e_2) \in S_{EX}(\tilde{E}) : s_{e_1,t}(\tilde{E}) + s_{e_2,t}(\tilde{E}) \leq 1 \quad \forall t \in T$$

- Fixed examinations:

$$\forall e \in E_{fix}(\tilde{E}) : s_{e,t}(\tilde{E}) = F_{e,t} \quad \forall t \in T$$

- Minor examinations:

$$\forall e \in E_{min}(\tilde{E}) : s_{e,t}(\tilde{E}) = 0 \quad \forall t \in T \setminus T_{min}$$

- Major examinations:

$$\forall e \in E_{maj}(\tilde{E}) : s_{e,t}(\tilde{E}) = 0 \quad \forall t \in T_{min}$$

- Unavailabilities:

$$\text{If } U_{e,t} = 1 \text{ for } (e, t) \in \tilde{E} \times T \text{ then: } s_{e,t}(\tilde{E}) = 0$$

All these constraints can be reduced to three constraints from the matrix  $AP$ :

- $\sum_{t \in T} s_{e,t}(\tilde{E}) = 1 \quad \forall e \in \tilde{E}$
- If  $AP_{e,t} = 0$  for  $(e, t) \in \tilde{E} \times T$  then:  $s_{e,t}(\tilde{E}) = 0$
- $\forall (e_1, e_2) \in S_{EX}(\tilde{E}) : s_{e_1,t}(\tilde{E}) + s_{e_2,t}(\tilde{E}) \leq 1 \quad \forall t \in T$

A subtlety we add to our model is the fact that we can build our partial schedule  $s(\tilde{E})$  on the basis of another partial schedule  $s(E_{basis})$  s.t  $E_{basis} \subseteq \tilde{E}$  given as input. Suppose that the partial schedule  $s(E_{basis})$  is given in a binary representation. In order to model this we add the constraints:

$$s_{e,t}(\tilde{E}) = s_{e,t}(E_{basis}) \quad \forall (e, t) \in E_{basis} \times T$$

This feature will be the basis of algorithms solving UCT-EPL in a more efficient way. If we do not use this feature, i.e  $E_{basis} = \emptyset$ , we will shorten the model notation from

$M_{ILP-GD}(\tilde{E}, \emptyset)$  to  $M_{ILP-GD}(\tilde{E})$ . The next step is to build the objective function that will compute  $P^{GD}$  for  $s(\tilde{E})$ . For this we first need to be able to count the number of  $Gap_{g_t, g_d}$  in the partial schedule  $n_{Gap_{g_t, g_d}}(s(\tilde{E}))$  we have for  $(g_t, g_d) \in S_{UGAP}$ . To do this we introduce the dummy variables  $c$  and a constraint to set it.

- $c \in \{0, 1\}^{|S_C(\tilde{E})| \times |S_{UGAP}|}$ .

$$c_{e_1, e_2, g_t, g_d} = \begin{cases} 1 & \text{if the courses } e_1 \text{ and } e_2 \text{ that are in conflict take places at time slots} \\ & \text{that are separated by } g_t \text{ time slot index(es) and } g_d \text{ day index(es)} \\ 0 & \text{else} \end{cases}$$

$$\forall (e_1, e_2, g_t, g_d) \in S_C(\tilde{E}) \times S_{UGAP}$$

- Set the variable  $c$ :

$$\forall (e_1, e_2, t_1, t_2) \in S_{UC}(\tilde{E})$$

$$s_{e_1, t_1}(\tilde{E}) + s_{e_2, t_2}(\tilde{E}) \leq 1 + c_{e_1, e_2, |I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|}$$

$$\begin{aligned} s^*(\tilde{E}) = \arg \min_{s, c} \quad & \sum_{(g_t, g_d) \in S_{UGAP}} pen_{g_t, g_d} \sum_{(e_1, e_2) \in S_C(\tilde{E})} c_{e_1, e_2, g_t, g_d} C_{e_1, e_2} \\ \text{subject to} \quad & \sum_{t \in T} s_{e, t}(\tilde{E}) = 1 \quad \forall e \in \tilde{E}, \\ & \forall (e, t) \in \tilde{E} \times T \text{ st } AP_{e, t} = 0 : s_{e, t}(\tilde{E}) = 0, \\ & \forall (e_1, e_2) \in S_{EX}(\tilde{E}) : s_{e_1, t}(\tilde{E}) + s_{e_2, t}(\tilde{E}) \leq 1 \quad \forall t \in T, \\ & s_{e, t}(\tilde{E}) = s_{e, t}(E_{basis}) \quad \forall (e, t) \in E_{basis} \times T \\ & \forall (e_1, e_2, t_1, t_2) \in S_{UC}(\tilde{E}) : \\ & s_{e_1, t_1}(\tilde{E}) + s_{e_2, t_2}(\tilde{E}) \leq 1 + c_{e_1, e_2, |I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|}, \\ & c \in \{0, 1\}^{|S_C(\tilde{E})| \times |S_{UGAP}|}, \\ & s(\tilde{E}) \in \{0, 1\}^{|\tilde{E}| \times |T|}. \end{aligned}$$

Table 4.1: UCT-EPL( $\tilde{E}$ ) ILP model:  $M_{ILP-GD}(\tilde{E}, s(E_{basis}))$

The number of  $Gap_{g_t, g_d}$ :  $n_{Gap_{g_t, g_d}}$  for  $(g_t, g_d) \in S_{UGAP}$  is then:

$$n_{Gap_{g_t, g_d}} = \sum_{(e_1, e_2) \in S_C(\tilde{E})} c_{e_1, e_2, g_t, g_d} C_{e_1, e_2}$$

and now we can express our penalty function  $P^{GD}$  as a linear objective function and

minimize it in our model to find the optimal partial schedule  $s^*(\tilde{E})$ :

$$s^*(\tilde{E}) = \arg \min_{s,c} \sum_{(g_t, g_d) \in S_{UGAP}} n_{Gap_{g_t, g_d}} pen_{g_t, g_d}$$

where  $pen$  was introduced in Section 3.3.1. The whole model is summarized in Table 4.1.

### 4.3.2 Integer Nonlinear Program: $M_{INP-BSR}(\tilde{E}, s(E_{basis}))$

In parallel to the linear binary program, we build a INP modeling UCT-EPL-BSR( $\tilde{E}$ ). Nonlinear programs allow more freedom in terms of modeling. For instance in this case, we can use  $P^{BSR}$ , which we established to be our main quantifier, as a nonlinear objective. On the other hand, these models sometimes cannot be solved by off the shelf solvers such as CPLEX<sup>®</sup> and require the use of heuristic methods.

Due to this freedom in modeling, the definition of  $s(\tilde{E})$  and the constraints are identical to their formulations in Section 4.2. The objective function to minimize is  $P^{BSR}$  defined in Section 3.3.2. The goal then is to find the optimal partial schedule  $s^*(\tilde{E})$ :

$$s^*(\tilde{E}) = \arg \min_{s(\tilde{E}) \in \mathcal{S}(\tilde{E})} P^{BSR}(s(\tilde{E}))$$

The complete model is given in Table 4.2.

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\min_s$   | $\sum_{v \in V} \sum_{t \in T} [(A_{v,t}(s(\tilde{E})) - A_{v,t}^{Lin}(s(\tilde{E}))) - (A_{v,t}^f(s(\tilde{E})) - A_{v,t}^{f,Lin}(s(\tilde{E})))]$                                                                                                                                                                                                                                                                                                                              |
| subject to | $\forall (e_1, e_2) \in S_{EX}(\tilde{E}) : s_{e_1}(\tilde{E}) \neq s_{e_2}(\tilde{E}),$<br>$\forall e \in E_{fix}(\tilde{E}) \forall t \in T \text{ s.t } F_{e,t} = 1 : s_e(\tilde{E}) = F_{e,t},$<br>$\forall e \in E_{min}(\tilde{E}) : s_e(\tilde{E}) \in T_{min},$<br>$\forall e \in E_{maj}(\tilde{E}) : s_e(\tilde{E}) \notin T_{min},$<br>$\forall (e, t) \in \tilde{E} \times T : s_e(\tilde{E}) \neq t \text{ if } U_{e,t} = 1$<br>$s(\tilde{E}) \in T^{ \tilde{E} }.$ |

Table 4.2: UCT-EPL( $\tilde{E}$ ) INP model:  $M_{INP-BSR}(\tilde{E}, s(E_{basis}))$

## 4.4 Problem decomposition

In order to obtain an optimal schedule of UCT-EPL-BSR we could solve  $M_{INP-BSR}(E)$ . Since it is a INP, there is not much hope to find a suitable algorithm returning the optimal schedule in a reasonable time. We could then resign ourselves to solve  $M_{ILP-GD}(E)$  using a traditional solver. Still, additionally to the fact that an optimal schedule for this model is not necessarily optimal for EPL-UCT-BSR, there is a risk that CPLEX<sup>®</sup> would not find an optimal schedule within a satisfactory time.

We could workaround these running time uncertainties by decomposing the problem. This implies the creation of several auxiliary optimization subproblems of reduced size from which it would be possible to generate the complete schedule. The auxiliary subproblems would be much more easier to solve than the initial problem and we could expect a gain in the running time.

We call *decomposition methods* schemes that are used to break down UCT-EPL into a series of optimization subproblems. We present two decomposition methods and we explain how to use them on UCT-EPL-BSR (resp. UCT-EPL-GD). The first decomposition method is general in the sense that it can be applied to any optimization model. On the other hand the second decomposition method is specific to UCT-EPL, i.e. it is based on his specific property.

### 4.4.1 Penalty avoidable courses decomposition method

Let us first introduce three observations that led to the idea of this decomposition method.

*Suppose we have an initial feasible schedule  $s(\tilde{E})$ . If we extract a partial schedule  $s(\tilde{E}_{sub})$  from this one then the penalty function can only decrease or stay the same, i.e  $P^{BSR}(s(\tilde{E}_{sub})) \leq P^{BSR}(s(\tilde{E}))$  (resp.  $P^{GD}(s(\tilde{E}_{sub})) \leq P^{GD}(s(\tilde{E}))$ ). Let  $s^*(\tilde{E})$  be the optimal partial schedule of UCT-EPL-BSR( $\tilde{E}$ ) (resp. UCT-EPL-GD( $\tilde{E}$ )). By extracting  $s(\tilde{E}_{sub})$  we obtain a partial feasible schedule of UCT-EPL-BSR( $\tilde{E}_{sub}$ ) (resp. UCT-EPL-GD( $\tilde{E}_{sub}$ )) and we observe thus that  $P^{BSR}(s^*(\tilde{E})) \geq P^{BSR}(s(\tilde{E}_{sub})) \geq P^{BSR}(s^*(\tilde{E}_{sub}))$  with  $s^*(\tilde{E}_{sub})$  being the optimal partial schedule of UCT-EPL-BSR( $\tilde{E}_{sub}$ ).*

*Suppose we have a partial feasible schedule  $s(\tilde{E})$ . If for this schedule, the courses in a subset  $\tilde{E}_{sub} \subset \tilde{E}$  are scheduled without generating any unwanted gaps, i.e  $\text{Gap}_{g_t, g_d}$  for  $(g_t, g_d) \in S_{UGAP}$ , then these courses have no impact on both penalty functions  $P^{GD}$  and  $P^{BSR}$ .*

*Suppose now there is one subset of courses  $E_{pa}$ , for which no matter how the other courses of the session in  $E_{ex} = E \setminus E_{pa}$  are scheduled, there is always a way of scheduling the courses*

in  $E_{pa}$  afterwards without generating any unwanted gaps. This subset  $E_{pa}$  is qualified as a **penalty avoidable subset of courses**. Suppose we have  $s^*(E_{ex})$  being an optimal partial schedule of UCT-EPL-BSR( $E_{ex}$ ) (resp. UCT-EPL-GD( $E_{ex}$ )). Let  $s(E_{pa})$  be a partial schedule for the courses in  $E_{pa}$  built such that  $s(E)$ , obtained by merging  $s^*(E_{ex})$  and  $s(E_{pa})$ , is a feasible schedule and the courses in  $E_{pa}$  do not generate any unwanted gaps in this schedule. We know that such a schedule exists since  $E_{pa}$  is a penalty avoidable subset of courses. We deduce from the previous observation that  $P^{BSR}(s(E)) = P^{BSR}(s^*(E_{ex}))$  (resp.  $P^{GD}(s(E)) = P^{GD}(s^*(E_{ex}))$ ) and from the first observation it follows that  $s(E)$  is an optimal schedule of UCT-EPL-BSR( $E$ ) (resp. UCT-EPL-GD( $E$ )).

These observations basically suggest that, having a penalty avoidable subset of courses  $E_{pa}$ , we can first focus on finding an optimal partial schedule  $s^*(E_{ex})$  of UCT-EPL-BSR( $E_{ex}$ ) (resp. UCT-EPL-GD( $E_{ex}$ )) and afterwards schedule the courses in  $E_{pa}$  such that they do not generate any unwanted gaps and lead to a feasible merged schedule.

We sketch now why using this decomposition might be interesting in our case. Solving the initial UCT-EPL-BSR( $E$ ) (resp. UCT-EPL-GD( $E$ )) would require to look at  $|T|^E$  schedules using exhaustive search. Suppose now we have a penalty avoidable subset of courses:  $E_{pa}$  and let  $E_{ex}$  be  $E \setminus E_{pa}$ . The total number of schedules to look at for the decomposition method to solve the UCT-EPL-BSR( $E_{ex}$ ) and schedule the courses in  $E_{pa}$  afterwards is in the worst case:  $|T|^{|E|-|E_{pa}|} + |T|^{|E_{pa}|}$ . Now since  $0 < |E_{pa}| < |E|$  and we make the assumption that  $|T| > 2$  it follows that  $|T|^{|E|-|E_{pa}|} + |T|^{|E_{pa}|} \leq |T|^E$  as proven in Appendix A. The minimum worst case complexity that can be achieved by the decomposition method is accomplished when extracting a subset  $E_{pa}$  of cardinality  $\frac{|E|}{2}$  as also shown in Appendix A. We will later explain how the algorithm that fixes the courses of  $E_{pa}$  afterwards has in practice much less partial schedules to enumerate for classical UCT-EPL instances. Constructing a subset  $E_{pa}$  with the greatest cardinality leads in practice to the best performances.

Before presenting the steps of this decomposition method more into details we will first:

- Define the notion of **dependence examinations set of a course**  $ex \in E$ :  $S_D^{ex}$ , **dependence examinations set of subset of courses**  $E_{sub} \subset E$ :  $S_D^{E_{sub}}$  and **dependence graph of a subset of courses**  $E_{sub} \subset E$ :  $G_D^{E_{sub}}$ . This will ease the tasks of finding a penalty avoidable subset of courses and of scheduling the courses of this subset afterwards without creating any unwanted gap.
- Explain how we can build the penalty avoidable subset of courses.
- Give the conditions that must be met so that a course of a penalty avoidable subset of courses can be scheduled and show how we find a time slot where it can be scheduled afterwards.

DEFINITION 19. The **dependence examinations set of a course**  $ex \in E$  is defined as the subset of course  $S_D^{ex} = \{e \in E \mid e \neq ex \wedge (C_{e,ex} > 0 \vee Excl_{e,ex} = 1)\} \subset E$ . The organization of the courses in the dependence examinations set of courses of  $ex$  and the particular administrative constraints on this course are the only constraints that will determine the allowed time slots where the course  $ex$  can be scheduled without generating unwanted gaps.

DEFINITION 20. The **dependence examinations set of subset of courses**  $E_{sub} \subset E$  is the extension of the dependence examinations set of a course concept to a subset of courses. It is defined as  $S_D^{E_{sub}} = \bigcup_{ex \in E_{sub}} S_D^{ex} \setminus E_{sub}$

DEFINITION 21. The **dependence graph of a subset of courses**  $E_{sub} \subset E$ :  $G_D^{E_{sub}} = (A, B)$  is a graph where each courses in  $E_{sub}$  has a corresponding vertex in A. For each course  $e_1 \in E_{sub}$  and for each course  $e_2 \in S_D^{e_1}$  there is an edge in B between corresponding vertexes of  $e_1$  and  $e_2$ .

We will now explain how to build a penalty avoidable subset of courses  $E_{pa}$ . A subset of courses  $E_{pa}$  is penalty avoidable if for any feasible partial schedule  $s(E_{ex})$ , with  $E_{ex} = E \setminus E_{pa}$  we can find at least a partial schedule  $s(E_{pa})$  such that in the schedule  $s(E)$  obtained by merging  $s(E_{ex})$  and  $s(E_{pa})$  there is no unwanted gaps. We have seen theoretically that we have to find a subset  $E_{pa}$  with cardinality close to  $\frac{|E|}{2}$  to have the minimum worst case number of schedule to check using the decomposition method. In the average case we will show that our algorithm that finds a partial feasible schedule for  $E_{pa}$  has a complexity that is much more reasonable than  $|T|^{E_{pa}}$  so we can make this subset much larger if possible. Though we will see later that we could only find  $E_{pa}$  that are quite small relative to  $E$  for classical instances of UCT-EPL. Two intuitive but very important properties of a penalty avoidable subset of courses that will helps us to build a bigger penalty avoidable subsets are:

- Let  $E_{pa_1}$  and  $E_{pa_2}$  be two penalty avoidable subsets then  $E_{pa_1} \cup E_{pa_2}$  is penalty avoidable. But its reverse implication is not true: if  $E_1 \cup E_2$  is penalty avoidable it does not necessarily imply that  $E_1$  and  $E_2$  are penalty avoidable subsets of courses.
- Let  $E_1 \cup E_2$  be a penalty avoidable subsets. If  $S_D^{E_1} \cap E_2 = \emptyset$  and symmetrically  $S_D^{E_2} \cap E_1 = \emptyset$  then  $E_1$  and  $E_2$  are penalty avoidable subsets.

A way to check if subset  $E_{sub} \subset E$  is a penalty avoidable subset of courses, is to verify that there is no feasible partial schedule  $s(S_D^{E_{sub}})$  such that we cannot find a partial schedule for the courses in  $E_{sub}$ . Let

- $S_{UC}^{ex,t_1} = \{(e, t_2) \in S_D^{ex} \times T \mid (e, ex, t_2, t_1) \in S_{UC} \vee (e, ex, t_1, t_2) \in S_{UC} \vee (ex, e, t_2, t_1) \in S_{UC} \vee (ex, e, t_1, t_2) \in S_{UC}\}$
- $S_{EX}^{ex} = \{e \in S_D^{ex} \mid (ex, e) \in S_{EX} \vee (e, ex) \in S_{EX}\}$

- $S_{FS}^{E_{pa}} = \{fs \in T^{|E_{pa}|} \mid \forall e \in E_{pa} : AP_{e,fs_e} \neq 0 \wedge \forall (e_1, e_2) \in S_C(E_{pa}) : (|I_{fs_{e_1}} - I_{fs_{e_2}}| > max_{gt} \vee |D_{fs_{e_1}} - D_{fs_{e_2}}| > max_{gd}) \wedge \forall (e_1, e_2) \in S_{EX}(E_{pa}) : fs_{e_1} \neq fs_{e_2}\}$
- $s \in \{0, 1\}^{|S_D^{ex} \cup \{ex\}| \times |T|}$ :

$$s_{ex,t} = \begin{cases} 1 & \text{if exam } ex \text{ can be assigned to time slot } t \\ & \text{while respecting hard constraints and} \\ & \text{without generating unwanted gaps} \\ 0 & \text{else} \end{cases} \quad \forall (ex, t) \in E_{pa} \times T$$

$$s_{e,t} = \begin{cases} 1 & \text{if exam } e \text{ is assigned to time slot } t \\ 0 & \text{else} \end{cases} \quad \forall (e, t) \in S_D^{E_{pa}} \times T$$

- $x \in \{0, 1\}^{|S_{FS}^{E_{pa}}|}$ :

$$x_{fs} = \begin{cases} 1 & \text{if merging } fs(E_{pa}) \text{ and } s(S_D^{E_{pa}}) \\ & \text{lead to a feasible schedule where courses in } E_{pa} \\ & \text{do not create unwanted gaps.} \\ 0 & \text{else} \end{cases} \quad \forall fs \times S_{FS}^{E_{pa}}$$

The linear integer programming of Table 4.3 searches for a feasible partial schedule of the courses in  $S_D^{E_{pa}}$  that minimizes the number of feasible partial schedules  $s(E_{pa})$  while not generating unwanted gaps. If the optimal value, i.e the number of feasible partial schedules  $s(E_{pa})$  that do not generate unwanted gaps, is greater than zero then the subset  $E_{pa}$  is penalty avoidable else it is not. Note that a solver like CPLEX<sup>®</sup> gives a lower cut value and as soon as the lower cut is greater than zero we can stop the resolution and deduct that this subset is not penalty avoidable. The worst case complexity of this linear integer programming model solved by enumeration is not good. For example there can be in the worst case  $|T|^{E_{pa}}$  binary variables  $x$ , leading to  $2^{|T|^{E_{pa}}}$  assignment of theses. So we can only apply it to low cardinality  $E_{pa}$ . For instance of UCT-EPL, it works quite well for  $|E_{pa}| \leq 3$  but it becomes intractable to solve in reasonable time for bigger cardinality. Now, instead of testing if every subsets formed of  $k$ -combinations of courses for  $k \in \{1, \dots, 3\}$  is penalty avoidable and building  $E_{pa}$  incrementally by merging the subsets, we use a smarter trick. For every subsets of courses  $E_{sub}$  formed from a  $k$ -combinations of courses we check that  $G_D^{E_{sub}}$  is connected. If it is not the case and suppose  $E_{sub}$  is a penalty avoidable subset of courses, the connected components of this graph verify the condition of the second property of penalty avoidable subsets of courses. Since the subset of courses in each connected component was already checked for lower value of  $k$  we do not have to check this  $k$ -combination again since if the subsets of the connected component are penalty avoidable they are already in  $E_{pa}$ .

$$\begin{aligned}
& \min_{x,s} \quad \sum_{fs \in S_{FS}^{E_{pa}}} x_{fs} \\
& \text{subject to} \quad \sum_{t \in T} s_{e,t} = 1 \quad \forall e \in S_D^{E_{pa}}, \\
& \quad \forall (e, t) \in (S_D^{E_{pa}} \cup E_{pa}) \times T \text{ st } AP_{e,t} = 0 : s_{e,t} = 0, \\
& \quad \forall (e_1, e_2, t) \in S_D^{E_{pa}} \times T : s_{e_1,t} + s_{e_2,t} \leq 1 \\
& \quad \forall (ex, t_1) \in E_{pa} \times T : 1 - \sum_{(e,t_2) \in S_{UC}^{ex,t_1}} s_{e,t_2} - \sum_{e \in S_{EX}^{ex}} s_{e,t_1} - AP_{ex,t_1} \leq s_{ex,t_1} \\
& \quad \forall fs \in S_{FS}^{E_{pa}} : 1 - \sum_{e_i \in E_{pa}} (1 - s_{e_i, fs_{e_i}}) \leq x_{fs} \\
& \quad s \in \{0, 1\}^{|S_D^{E_{pa}} \cup E_{pa}| \times |T|}. \\
& \quad x \in \{0, 1\}^{|S_{FS}^{E_{pa}}|}.
\end{aligned}$$

Table 4.3: Linear integer programming model to identify penalty avoidable subset of courses

Now we can move on the explanations of how we can schedule the courses in  $E_{pa}$  afterwards, based on a partial feasible schedule  $s(E \setminus E_{pa})$ . In order to find a time slot where a course  $e_1 \in E_{pa}$  can be scheduled afterwards without generating unwanted gaps we only need to know a partial schedule  $s(S_D^{e_1})$ . Since there can be other courses  $e_2 \in E_{pa} \cap S_D^{e_1}$  that are not already fixed, we need to fix these both courses together. We can find the set of courses that need to be scheduled together as the courses whose vertexes belong to the same connected components of the dependence graph  $G_D^{E_{pa}}$ . For each subset of courses  $E_{pa}^k$  corresponding to vertexes of the k-th found connected component in the dependence graph we solve the constraint satisfaction problem of Table 4.4 to find a partial schedule  $s(E_{pa}^k)$  using partial feasible  $s(S_D^{E_{pa}^k})$  extracted from  $s(E \setminus E_{pa})$ . Suppose we have n connected components. The total number of partial schedule to look at in the worst case in order to find a feasible partial schedule for  $s(E_{pa})$  went down from  $|T|^{E_{pa}}$  to  $n|T|^{E_{pa}^q}$  where q is such that  $|E_{pa}^q| \geq |E_{pa}^k|$  for all  $k = \{1, \dots, n\}$ . In classical instance of the EPL-UCT problem the number of connected components is quite big w.r.t to  $|E_{pa}|$  and  $|E_{pa}^q|$  is small this is the reason why we should make  $E_{pa}$  as big as possible.

$$\begin{array}{ll}
\min_s & 0 \\
\text{subject to} & \sum_{t \in T} s_{ex,t} = 1 \quad \forall ex \in E_{pa}^k \\
& \forall t \in T \text{ st } AP_{ex,t} = 0 : s_{ex,t} = 0, \quad \forall ex \in E_{pa}^k \\
& \forall (t_1, e, t_2) \in T \times S_{UC}^{ex,t_1} : s_{ex,t_1} \leq 1 - s_{e,t_2} \\
& \forall e \in S_{EX}^{ex} : s_{ex,t_1} \leq 1 - s_{e,t_1}(S_D^{ex}) \\
& \forall (e, t) \in S_D^{E_{pa}^k} \times T : s_{e,t} = s_{e,t}(S_D^{E_{pa}^k}) \\
& s \in \{0, 1\}^{(E_{pa}^k \cup S_D^{E_{pa}^k}) \times |T|}.
\end{array}$$

Table 4.4: Constraint satisfaction problem to schedule the set of courses in  $E_{pa}^k$ 

We have now all the tools to present the steps of the decomposition method. Firstly we built a maximum cardinality penalty avoid subset of courses  $E_{pa}$  with the earlier described methodology. After transforming the UCT-EPL-BSR( $E$ ) (resp. UCT-EPL-GD( $E$ )) into two auxiliary subproblems that can be solved following those steps :

1. For  $E_{ex} = \tilde{E} \setminus E_{pa}$ . Find  $s^*(E_{ex})$ , i.e the optimal schedule of UCT-EPL-BSR( $E_{ex}$ ) (resp. UCT-EPL-GD( $E$ )) problem, by solving  $M_{INP-BSR}(E_{ex})$  (resp.  $M_{ILP-GD}(E_{ex})$ ).
2. Obtain the different connected components of the dependence graph of  $E_{pa}$ . Let  $n_{CC}$  be the number of obtained connected components. Solve for each subset of courses  $E_{pa}^k$  corresponding to nodes of the  $k$ -th connected component  $CC_k$  the constraint satisfaction problem of Table 4.4 using partial feasible schedule  $s^*(E_{ex} \cap S_D^{E_{pa}^k})$ . Let  $s_{E_{pa}^k}$  be the partial feasible schedule found for each  $k \in \{1, \dots, n_{CC}\}$  then by merging those partial schedules and  $s^*(E_{ex})$ , we obtain an optimal schedule to UCT-EPL-BSR( $E$ ) (resp. UCT-EPL-GD( $E$ )).

#### 4.4.2 Independent subproblems decomposition

This method is based on breaking down an optimization model in a series of independent optimization sub-models. This method can be applied to linear and nonlinear programs.

Let the following optimization model be as general as possible:

$$\begin{array}{ll}
z = \min_x & f(x) \\
\text{subject to} & g(x) = 0
\end{array}$$

where

- $m$  is the number of constraints.
- $n$  is the number of decision variables.
- $x \in \mathbb{R}^n$  is a vector of decision variables.
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is a function (nonlinear or linear).
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is a vector function (nonlinear or linear).

Suppose now we can rewrite it as:

$$\begin{aligned}
 z = \min_x \quad & \sum_{k=1}^q f_k(x_{i_{k1}}, \dots, x_{i_{kn_k}}) \\
 \text{subject to} \quad & g_1(x_{i_{11}}, \dots, x_{i_{1n_1}}) = 0 \\
 & \dots \\
 & g_q(x_{i_{q1}}, \dots, x_{i_{qn_q}}) = 0
 \end{aligned} \tag{4.5}$$

where

- $m$  is the number of constraints.
- $n$  is the number of decision variables.
- $x \in \mathbb{R}^n$  is a vector of decision variables.
- - $S = \{1, \dots, n\}$ .
  - $\forall k \in \{1, \dots, q\} : S_k \neq \emptyset$ .
  - $\forall k \in \{1, \dots, q\} : S_k \subset S$ .
  - $\bigcup_{k=1}^q S_k = S$
  - $\forall (k_1, k_2) \in \{1, \dots, q\}^2, k_1 \neq k_2 : S_{k_1} \cap S_{k_2} = \emptyset$ .
  - $\forall (k, j) \in \{1, \dots, q\} \times \{1, \dots, n_k\} : i_{kj} \in S_k$ .
- $f_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}$  is a function (nonlinear or linear)
- $g_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{m_k}$  is a vector function (nonlinear or linear)

Then we can separate the initial problem in  $q$  subproblems and merge their optimal solution to get the optimal solution of the initial problem. Formally we have:

$$z = \sum_{i=1}^q z_k$$

with

$$\begin{aligned}
 z_k = \min_{x_{i_{k1}}, \dots, x_{i_{kn_k}}} \quad & f_k(x_{i_{k1}}, \dots, x_{i_{kn_k}}) \\
 \text{subject to} \quad & g_k(x_{i_{k1}}, \dots, x_{i_{kn_k}}) = 0
 \end{aligned}$$

for  $k \in \{1, \dots, q\}$ .

It is easy for a linear integer program such as  $M_{ILP-GD}$  given in Table 4.1 to be rearranged like the optimization program 4.5. From the primal constraint graph we extract the connected components. Let us suppose we have  $q$  connected components. Let  $CC_k$  be the set of vertexes corresponding to the  $k$ -th connected component for  $k \in \{1, \dots, q\}$ . We build  $S_k$  such that it contains the indexes of the variables corresponding to the vertexes of  $CC_k$ . Then to build  $g_k$  we take all constraint where one of the variables of indexes contained in  $S_k$  is present. We build  $f_k$  from the terms of the objective function containing a variable with index contained in  $S_k$ . This technique is illustrated in Example 3.

EXAMPLE 3.

$$\begin{aligned} z &= \min_{x_1, x_2, x_3, x_4} && 3x_1 + 4x_2 + x_3 + 5x_4 \\ \text{subject to} &&& x_1 < 2x_2, \\ &&& x_3 \geq x_4, \\ &&& x_1, x_2, x_3, x_4 \in \{0, 1\}. \end{aligned}$$

For this integer binary program, we have 2 connected components:  $CC_1 = \{x_1, x_2\}$  and  $CC_2 = \{x_3, x_4\}$  and the initial problem can be transformed to:

$$z = z_1 + z_2$$

with

$$\begin{aligned} z_1 &= \min_{x_1, x_2} && 3x_1 + 4x_2 \\ \text{subject to} &&& x_1 < 2x_2, \\ &&& x_1, x_2 \in \{0, 1\}. \end{aligned}$$

and

$$\begin{aligned} z_2 &= \min_{x_3, x_4} && x_3 + 5x_4 \\ \text{subject to} &&& x_3 \geq x_4, \\ &&& x_3, x_4 \in \{0, 1\}. \end{aligned}$$

In Example 4, we sketch how this decomposition method can be important in term of complexity reduction with a toy problem.

EXAMPLE 4. Suppose we have a initial optimization model with  $n$  binary variables and that our solving algorithm is simply an exhaustive search. If we consider that generating a solution, checking if it is feasible and comparing it to the best feasible solution obtained so far as a unique operation then our exhaustive search is  $\mathcal{O}(2^n)$ .

Suppose now can decompose the original optimization subproblem in  $q = \frac{n}{c}$  independent optimization subproblems each of them containing  $c$  binary variables with  $c \in \mathbb{N}_0$ . Then based on the same assumption as in the previous paragraph, finding an optimal solution for each independent subproblem is at most  $\mathcal{O}(2^c)$  work. Hence the total work to obtain a optimal solution of the initial model is  $\mathcal{O}(\frac{n2^c}{c})$  which is linear in  $n$ . If an operation takes one nanosecond, one can observe in Table 4.8 the effect on the complexity of the independent subproblems decomposition methods for different pairs of values of  $c$  and  $n$ .

| $n \backslash c$ | 1                           | 5                         | 20                     |
|------------------|-----------------------------|---------------------------|------------------------|
| 20               | 0.001 sec                   | $16 \cdot 10^{-9}$ sec    | $2 \cdot 10^{-9}$ sec  |
| 100              | $4 \cdot 10^{13}$ years     | 0.001 sec                 | $32 \cdot 10^{-9}$ sec |
| 500              | $1.03 \cdot 10^{134}$ years | $2 \cdot 10^{14}$ years   | 0.67 sec               |
| 1000             | $3.4 \cdot 10^{284}$ years  | $2.5 \cdot 10^{44}$ years | 0.7140 year            |

Table 4.5: Impact on the running time of the independent subproblems decomposition methods for different values of  $c$  and  $n$ .

### 4.4.3 Nested decomposition technique

The idea we will present in this subsection is that the two decomposition techniques explained previously can be used jointly for UCT-EPL-BSR( $E$ ) problem or for UCT-EPL-GD( $E$ ). We nest ideas of both methods in order to create subproblems whose resolution is less complex than the initial problem.

We first apply the algorithm to find a penalty avoidable subset of courses  $E_{pa}$ . Find the subsets of courses  $E_{pa}^k$  for  $k \in \{1, \dots, n\}$  corresponding to the vertexes in the  $k$ -th connected components. Then we solve for each of these subsets of courses  $E_{pa}^k$  the constraint satisfaction problem of Table 4.4 as soon as all the courses in  $S_D^{E_{pa}^k}$  are scheduled. Note that these constraint satisfaction problem can be solved independently and concurrently. Then we construct  $M_{INP-BSR}(E_{ex})$  (resp.  $M_{ILP-GD}(E_{ex})$ ) to solve

UCT-EPL-BSR( $E_{ex}$ ) (resp. UCT-EPL-GD( $E_{ex}$ )). We break this model using the independent subproblems decomposition method and we can solve each of the models created directly, independently and concurrently.

## 4.5 Invariants of the gaps distribution

In the first approach of the KPI's, we characterized a good schedule as one that has a gaps distribution with  $Gap_{g_t, g_d}$  as high as possible for large lexicographical pairs  $(g_t, g_d)$  and then exponentially decreasing with  $(g_t, g_d)$  for  $(g_t, g_d) \in S_{UGAP}$ . An interesting question is: "What are the characteristics of the distribution that we cannot affect?". For UCT-EPL we will present two characteristics: the quantity of  $Gap_{g_t, g_d}$  that are forced by administrative constraints for  $(g_t, g_d) \in S_{UGAP}$  and the minimal sum of gaps for a subset of  $S_{UGAP}$ . We detail for each of them the motivations that led to their study and how to compute them. For the second characteristic we will introduce a second weaker technique to compute it based on the notion of cliques in a graph having the advantage to be applicable to any UCT.

### 4.5.1 Minimal forced number of $Gap_{g_t, g_d}$

**Motivation** Administrative specifications sometimes enforce unavoidable gaps. It happens because of fixed courses, unavailabilities, oral duplicates that cannot be placed at the same time slot and minor and major exams that are to be placed at special weeks. We can learn from this characteristic that no matter how we schedule our examination session, we will have this quantity of  $Gap_{g_t, g_d}$  for  $(g_t, g_d) \in S_{UGAP}$ .

$$\begin{aligned}
 n_{Gap_{g_t, g_d}}^{min} = \min_{x, c} \quad & \sum_{(e_1, e_2) \in S_C} c_{e_1, e_2, g_t, g_d} C_{e_1, e_2} \\
 \text{subject to} \quad & \sum_{t \in T} x_{e, t} = 1 \quad \forall e \in E, \\
 & \forall (e, t) \in E \times T \text{ st } AP_{it} = 0 : x_{it} = 0, \\
 & \forall (e_1, e_2) \in S_{EX} : x_{e_1, t} + x_{e_2, t} \leq 1 \quad \forall t \in T, \\
 & \forall (e_1, e_2, t_1, t_2) \in S_{UC} : x_{e_1, t_1} + x_{e_2, t_2} \leq 1 + c_{e_1, e_2, |I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|}, \\
 & c \in \{0, 1\}^{|S_C| \times |S_{UGAP}|}, \\
 & x \in \{0, 1\}^{|E| \times |T|}.
 \end{aligned}$$

Table 4.6: ILP model to compute the minimal forced quantity of  $Gap_{g_t, g_d}$ .

**Computation** The model we solve to compute the quantity of minimal forced  $Gap_{g_t, g_d} : n_{Gap_{g_t, g_d}}^{min}$  for  $(g_t, g_d) \in S_{UGAP}$  is in Table 4.6.

### 4.5.2 Minimal quantity of gaps of a subset of $S_{UGAP}$

**Motivation** The previous characteristic was the minimal forced quantity for each gap individually. An extension of the first characteristic we are interested in is to compute the forced quantity of gaps for a subset  $S_{UGAP}^{sub} \subseteq S_{UGAP} : n_{S_{UGAP}^{sub}}^{min}$ . This quantity is computed as

$$n_{S_{UGAP}^{sub}}^{min} = \min_{s \in S} \sum_{(g_t, g_d) \in S_{UGAP}^{sub}} n_{Gap_{g_t, g_d}}(s)$$

We deduce that any schedule will have at least this quantity of gaps for the subset. By changing the weight of  $P^{GD}$ , we will end up with at least this quantity of unwanted gaps and we will only be able to shape a specific distribution on this subset.

**Computation** The model we solve to compute the forced quantity of gap numbers for the subset  $S_{UGAP}^{sub} : n_{S_{UGAP}^{sub}}^{min}$  is in Table 4.7.

$$\begin{aligned}
 n_{S_{UGAP}^{sub}}^{min} = \min_{x, c} \quad & \sum_{(g_t, g_d) \in S_{UGAP}^{sub}} \sum_{(e_1, e_2) \in S_C} c_{e_1, e_2, g_t, g_d} C_{e_1, e_2} \\
 \text{subject to} \quad & \sum_{t \in T} x_{e, t} = 1 \quad \forall e \in E, \\
 & \forall (e, t) \in E \times T \text{ st } AP_{e, t} = 0 : x_{e, t} = 0, \\
 & \forall (e_1, e_2) \in S_{EX} : x_{e_1, t} + x_{e_2, t} \leq 1 \quad \forall t \in T, \\
 & \forall (e_1, e_2, t_1, t_2) \in S_{UC} : x_{e_1, t_1} + x_{e_2, t_2} \leq 1 + c_{e_1, e_2, |I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|}, \\
 & c \in \{0, 1\}^{|S_C| \times |S_{UGAP}|}, \\
 & x \in \{0, 1\}^{|E| \times |T|}.
 \end{aligned}$$

Table 4.7: ILP model to compute the minimum quantity of gaps for a subset of  $S_{UGAP}$

### 4.5.3 Clique induced gaps

**Motivation** Based on cliques of courses found in the conflict graph, we can also deduce the imposed quantity of gaps for a subset of  $S_{UGAP}$ . The method we present here only provides lower bounds on the minimal quantity of gaps achievable for the subset while the previous model would exactly give this minimal quantity. The method we introduce has the advantages that it can be applied to any UCT problem without relying on a certain administrative system. It is also less heavy computationally if finding cliques in the conflict graph of the UCT problem is easy. It is actually the case

for standard instances of UCT-EPL where the maximal clique size is not that big and the conflict matrix is quite sparse.

**Computation** To present the way we compute this lower bound on the gaps numbers, we first introduce the procedure to build the conflict graph of UCT-EPL. We also introduce an auxiliary problem to calculate the maximal clique size inducing no gaps in  $S_{UGAP}^{(g_{mt}, g_{md})}$ . We denote it as  $MC(g_{mt}, g_{md})$ .

The definition of conflict graph was introduced in the theoretical background chapter. The procedure to obtain the conflict graph of the UCT-EPL( $\tilde{E}$ ) problem is the following:

Let  $G = (A, B)$  denotes the corresponding undirected conflict graph where  $A$  is the set of nodes of the graph and  $B$  is the set of vertices. These two sets are build in the following way from the parameters:

- For each course  $e \in \tilde{E}$  let us define a corresponding node  $e' \in A$ . Let  $map : \tilde{E} \rightarrow A$  be the mapping function that makes the examination-node correspondence.
- For each  $(e_1, e_2) \in S_C(\tilde{E})$  and  $e'_1 = map(e_1), e'_2 = map(e_2)$ , let us define a vertex  $(e'_1, e'_2) \in B$ . We introduce also the weight function  $w : B \rightarrow \mathbb{R}$  as  $w(e'_1, e'_2) = C_{e_1, e_2}$ .

We now introduce the  $MC(g_{mt}, g_{md})$  problem. Suppose we have a set of examinations to schedule with any pair of distinct courses  $e_1$  and  $e_2$  in conflict, i.e  $C_{i,j} > 0$ . We relax the hard constraints, meaning that the courses can be placed at any time slot. The subgraph of the conflict graph  $G$  induced by the set of vertices corresponding to these examinations is a clique. We search now for the maximum cardinality of such a set of examinations, i.e. a clique size, such that we can assign courses to time slots in a way that ensures  $n_{Gap_{g_t, g_d}} = 0 \forall (g_t, g_d) \in S_{UGAP}^{(g_{mt}, g_{md})}$  and such that increasing the size of the set would create a  $Gap_{g_t, g_d}$ .

Now that the problem is well defined let see how to actually compute the induced gaps. Suppose we have a clique  $Cl$  of size  $n$ . Let  $(s_{g_t}, s_{g_d})$  be the smallest lexicographical pair in  $S_{UGAP}$  such that the solution of the  $MC(g_{mt}, g_{md})$  problem is  $s_{MC}^*$  with  $s_{MC}^* < n$ . We know that no matter how we schedule the courses of the clique we will end up with at least a minimal quantity of

$$\min\{C_{e_1, e_2} | (e_1, e_2) \in S_C \wedge map(e_1), map(e_2) \in Cl\}$$

gaps in the subset  $S_{UGAP}^{(s_{g_t}, s_{g_d})}$ . This follows from the fact that we could in the best case only schedule a clique of cardinality  $s_{MC}^*$  without generating these gaps. Now suppose we do not account anymore for the two courses that created the gaps. We can reiterate the procedure since the subset created by removing these two courses forms a clique of size  $n - 2$  and it will also generate a forced quantity of gaps when they will be

scheduled. Since we cannot presume precisely which two courses will generate the gaps we continue with a size of clique  $n - 2$  but we keep looking for

$$\min\{C_{e_1,e_2} | (e_1, e_2) \in S_C \wedge \text{map}(e_1), \text{map}(e_2) \in Cl\}$$

to give a quantity of gaps generated by this reduced clique.

Now suppose we want to apply the procedure to all the cliques of  $G$  to be able to deduce a greater quantity of imposed gaps. We cannot apply the procedure to each clique of  $G$  individually since two cliques can share pairs of courses and thus we would count imposed gaps multiple times.

What we can do to solve the previous problem is to select a set of cliques where each pair of cliques have at most one node in common. By doing this we assure that no pair of courses in conflict will generate a quantity of imposed gap that are double counted. To build the set of cliques we proceed greedily. We search for the maximal clique, compute the quantity of imposed gaps, and carry on recursively with the next biggest clique that has at most one node in common with the set of cliques already found until we reach a minimal limit of clique size  $lim$ . Let  $s_{MC}^*$  be the solution to  $MC(max_{gt}, max_{gd})$ , then  $lim = s_{MC}^* + 1$ . If there is more than one clique of the same size, then we select the clique  $Cl_i$  that has the maximal  $\min\{C_{e_1,e_2} | (e_1, e_2) \in S_C \wedge \text{map}(e_1), \text{map}(e_2) \in Cl_i\}$ .

## 4.6 Symmetry breaking

Another method to reduce the size of the feasible solutions set of a combinatorial problem is to break symmetries. We will not go into details of the symmetry theory. Instead we will just give the intuition about how we can apply it to one of our earlier presented model.

$M_{ILP-GD}$  given in Table 4.1 contain what we call a variable symmetry. A variable symmetry is the existence of permutations of the variables, i.e. a bijection of the variables set on himself, that leads to the same optimal value for any solution of a model. Suppose there exist two courses  $e_1, e_2 \in E$  that satisfy the following conditions:

- $AP_{e_1,t} = AP_{e_2,t} \quad \forall t \in T$
- $\forall e_3 \in E, e_3 \neq e_1, e_2 : Excl_{e_1,e_3} = Excl_{e_2,e_3}$
- $\forall e_3 \in E, k \neq i, j : C_{e_1,e_3} = C_{e_2,e_3}$

Let  $s^*(E)$  be a solution to  $M_{ILP-GD}(E)$ . It is clear that if we permute the variables of  $s$  corresponding to courses  $e_1$  and  $e_2$  in the model for each time slot, i.e  $s_{e_1,t} \leftrightarrow s_{e_2,t} \quad \forall t \in T$  we would obtain the same optimum for this modified schedule.

In order to break this symmetry between the two courses we can add a precedence constraint between the two courses, i.e:

$$\forall t_1, t_2 \in T \text{ with } t_1 > t_2 : s_{e_1, t_1} + s_{e_2, t_2} \leq 1$$

such that the course  $e_1$  is never scheduled after the course  $e_2$  in the examination session.

We can easily generalize this scheme to a set of courses where any pair of courses satisfy the conditions. We could ask us by how much the symmetry breaking scheme reduce the number of feasible schedules. To sketch it we momentarily omit all the administrative constraints. Suppose we have a total of  $n$  courses where any pair of courses satisfy the conditions. We would initially be able to form  $|T|^n$  schedule. After having introduce the precedence constraints, the number of schedule is equal to the total number of  $n$ -combinations with repetitions, i.e the number of sequence of  $n$  elements from  $T$  allowing for duplicates but disregarding different orderings (e.g.  $(3,5,5) = (5,3,5)$ ). This number can be computed as  $\frac{(n+|T|-1)!}{n! (|T|-1)!}$ . The ratio of the number of schedules before symmetry breaking is given for different values of  $n$  and  $|T|$ . We can observe that when  $n$  and/or  $|T|$  increase(s) the ratio also increases. Table 4.8 shows the ratio of the numbers for different values of  $n$  and  $|T|$ . These are given under the form:  $\frac{\text{\# of schedules before decomposition}}{\text{\# of schedules after decomposition}} = \text{Ratio}$ . All the numbers are rounded with only one decimal in scientific representation.

|     |    | $ T $                                                  |                                                              |
|-----|----|--------------------------------------------------------|--------------------------------------------------------------|
|     |    | 5                                                      | 10                                                           |
| $n$ | 10 | $\frac{9.7 \cdot 10^7}{1001} = 9.7 \cdot 10^3$         | $\frac{1 \cdot 10^{10}}{92378} = 1.1 \cdot 10^5$             |
|     | 20 | $\frac{9.5 \cdot 10^{13}}{10626} = 9 \cdot 10^9$       | $\frac{1 \cdot 10^{20}}{1 \cdot 10^7} = 1 \cdot 10^{13}$     |
|     | 40 | $\frac{9.1 \cdot 10^{27}}{135751} = 6.7 \cdot 10^{22}$ | $\frac{1 \cdot 10^{40}}{2.1 \cdot 10^9} = 4.9 \cdot 10^{30}$ |
|     |    |                                                        |                                                              |

Table 4.8: Numbers of schedules before and after symmetry breaking

## 4.7 Extra instance properties

In this section we present other properties that characterize an instance of UCT-EPL.

### 4.7.1 Maximal general gap under non fixed courses

**Motivation** We define the maximal general gap under non fixed courses  $max_{gap}^{gen}$  as the maximal lexicographical gap that can be assured between any conflict pair of non-fixed examinations. More formally,  $(m_{g_t}^*, m_{g_d}^*) = max_{gap}^{gen} = \max_{lex}(g_t, g_d) \in S_{UGAP}$  such that

there exists a feasible schedule  $s$  that satisfies :

$$n_{S_{UGAP}(m_{g_t}^*, m_{g_d}^*)}^{min} - \sum_{(g_t, g_d) \in S_{UGAP}(m_{g_t}^*, m_{g_d}^*)} n_{Gap_{g_t, g_d}}^{min} = 0$$

If we can find a maximal general gap that is lexicographically close or greater than  $(max_{g_t}, max_{g_d})$  we know then that we will be able to build a good session for every student quite easily. In this case, it is likely that an optimal partial schedule of UCT-EPL-GD( $\tilde{E}$ ) will be optimal or near optimal to UCT-EPL-BSR( $\tilde{E}$ ). In other words, in this case it can be smarter to solve  $M_{ILP-GD}(\tilde{E})$  that will find very good partial schedules to UCT-EPL-BSR( $\tilde{E}$ ).

**Computation** It can be computed using the model of Table 4.7. We iteratively increase the subset by the next lexicographical gap beginning from  $(0, 0)$  and we subtract the sum of the minimal forced number of gaps for each  $Gap_{g_t, g_d}$  of this subset until the result is not zero. The maximal lexicographical gap that can be assured is then the gap added in the last step.

#### 4.7.2 Lexicographical optimization on $Gap_{g_t, g_d}$

$$\begin{aligned} n_{Gap_{g_t, g_d}}^{lex} = \min_{s, c} & \sum_{(e_1, e_2) \in S_C} c_{e_1, e_2, g_t, g_d} C_{e_1, e_2} \\ \text{subject to} & \sum_{t \in T} s_{e, t} = 1 \quad \forall e \in E, \\ & \forall (e, t) \in E \times T \text{ st } AP_{e, t} = 0 : s_{e, t} = 0, \\ & \forall (e_1, e_2) \in S_{EX} : s_{e_1, t} + x_{e_2, t} \leq 1 \quad \forall t \in T, \\ & \forall (e_1, e_2, t_1, t_2) \in S_{UC} : \\ & s_{e_1, t_1} + s_{e_2, t_2} \leq 1 + c_{e_1, e_2, |I_{t_1} - I_{t_2}|, |D_{t_1} - D_{t_2}|}, \\ & \forall (m, n) \in S_{UGAP} \text{ s.t } (m, n) \leq_{lex} (g_t, g_d) : \\ & \sum_{(e_1, e_2) \in S_C} c_{e_1, e_2, m, n} C_{e_1, e_2} = n_{Gap_{m, n}}^{lex} \\ & c \in \{0, 1\}^{|S_C| \times |S_{UGAP}|}, s \in \{0, 1\}^{|E| \times |T|}. \end{aligned}$$

Table 4.9: Linear integer program to optimize lexicographically the number of  $Gap_{g_t, g_d}$  for  $(g_t, g_d) \in S_{UGAP}$

**Motivation** In Section 3.3 , we presented different approaches to compare schedules based on gap's KPI. We justified the choice of a weighted objective for the first approach.

Here instead of weighting the gaps, we will study the impact of a lexicographical optimization of them on the distribution of the gaps.

**Computation** The model we solve to obtain the number of  $Gap_{g_t, g_d}$ :  $n_{Gap_{g_t, g_d}}^{lex}$  when we minimize lexicographically the number of  $Gap_{g_t, g_d}$  for  $(g_t, g_d) \in S_{UGAP}$  is in Table 4.9.

## 4.8 Summary

Considering that we are able to fully model UCT-EPL in two distinct ways, we can elaborate algorithms solving those models and combine them to produce a solver. Moreover the decomposition methods and invariants given in this chapter may be used to better assess or to improve the performances. Note that when the models computing the invariants are not solved to optimality, they can still provide interesting lower bounds.



# SOLVER DESCRIPTION

---

The general architecture of our solver combines three distinguishable main steps from data acquisition to the final schedule generation. The first one is the pre-optimization phase during which the pre-processing of the oral courses is performed. The following step is the optimization phase, involving successively three algorithms, to finish with a small post-optimization phase refining the solution.

These 3 phases and their respective algorithms are fully described in the present chapter. At the end, a section gives the full definition of our solver by explaining how the different algorithms successively interact with each other.

## 5.1 Pre-optimization phase

The 2 steps of the pre-processing are described below.

### 5.1.1 Oral examination duplication and optimal students assignments

**Duplication** Each oral examination is given with the capacity of students that can be interrogated during a time slot. As explained in the problem statement, for the convenience of the teacher, each oral examination is duplicated a minimum number of times such that all the enrolled students can take it. Mathematically, let  $cap_e$  denote the capacity of a course  $e$  and  $\#stud_e$  being the number of students enrolled in this course. The number of duplicates for exam  $e$  is then  $\left\lceil \frac{\#stud_e}{cap_e} \right\rceil$ .

**Assignment of students to duplicates** We chose to assign students to a duplicate before the optimization. If this assignment had to be made within the general optimization model, the size of the problem would grow significantly and it would become even more intractable for our methods to solve it. We assign students to oral duplicates by minimizing the total number of conflicting pairs of courses. A pair of courses is said in conflict when at least one student follows both courses. By doing so we hope to

reduce the number of variables for UCT-EPL( $\tilde{E}$ ) models, in comparison of a random assignment to orals. For example in model of Table 4.1, each removed conflicting pair of courses will reduce the size of  $S_C$  and thus the number of variables counting the conflicts (variables  $c$  in the model).

## 5.2 Optimization phase

In this section, we will explain the optimization algorithms we developed to solve the models of Section 4.3. The first two try to solve the  $M_{ILP-GD}$  using CPLEX<sup>®</sup> with a maximal given time. One algorithm is designed to solve  $M_{ILP-GD}(E)$  at once while the other apply a fix and solve method with several iterations. The last and third is designed to solve  $M_{INP-BSR}(E)$  using meta-heuristics. The optimization phase will combine these three algorithms in the finest possible proportions in order to generate the best possible schedule of UCT-EPL-BSR.

### 5.2.1 First algorithm: CPLEX<sup>®</sup> on Integer Linear Programming model

The first algorithm is quite simple. It solves  $M_{ILP-GD}(E)$  presented in Table 4.1 using CPLEX<sup>®</sup>. It is also possible to feed the algorithm with a good initial solution, given as MIP start to CPLEX<sup>®</sup>, that can speed up the resolution considerably. We add a maximal time stopping criterion since it is possible that CPLEX<sup>®</sup> could not find the optimal solution in a reasonable time because of the size of the model. This algorithm will be further referred as the CPLEX<sup>®</sup> on Integer Linear Programming model (CILP).

We will in fact show in the test chapter that this algorithm does not even reach a solution with small relative MIP gap value in a reasonable time, even for little sub-instances of real UCT-EPL data sets. This is why we came up with a second algorithm built on the same model but that tries to solve it by tackling subproblems of smaller size in a intelligent and adaptive way.

### 5.2.2 Second algorithm: Fix And Solve

Since  $M_{ILP-GD}(\tilde{E})$  is highly combinatorial even for small complicated sub-instances (see test in Section 6.4) there is almost no hope to solve it to optimality using the first algorithm. This is due to the fact that the search tree produced by the Branch & Bound is too dense and there is no chance to visit some parts of the tree.

Suppose we have a schedule  $s$ . We could consider keeping part of this schedule unchanged for a set of courses called the **fragment** (denoted  $frag$ ) and we optimized the rest of the courses that did not belong to this set with  $M_{ILP-GD}(E, s(frag))$  of Table

4.1, where  $s(frag)$  is extracted from  $s$ , with CPLEX<sup>®</sup>. This procedure would have the following implications:

- The size of the model (number of variables) is reduced since the courses of the fragment are fixed.
- Suppose  $s^*$  is the optimal schedule to  $M_{ILP-GD}(E, s(frag))$ , then  $P^{GD}(s^*) \leq P^{GD}(s)$ .
- If the courses of the fragment are not scheduled as in an optimal solution of  $M_{ILP-GD}(E)$  then the optimal value of  $M_{ILP-GD}(E, s(frag))$  is not a global minimum of  $M_{ILP-GD}(E)$ .

This means that if we find a size of fragment such that the reduced model can be solved in a reasonable time then we can build an iterative process that leads to better and better schedules of UCT-EPL-GD( $E$ ). The downside is that the schedule obtained at the end of the iteration process can be suboptimal.

We will call the iterative algorithm based on the idea of the previous paragraph: **Fix And Solve** (FAS) in the scope of this Thesis. This iterative method consists of selecting a set of variables, called the fragment, that are fixed to their value in the current solution and optimize again with a reduced size model to build an improved new solution. We designed the FAS algorithm to solve linear integer programming based on ideas taken from Large Neighborhood Search algorithm that solve constraint programming problems [Bes07].

We first introduce the parameters of the FAS algorithm and then explain the reasons that led to the final design of these ones:

- $time$  is the total time allowed to the algorithm
- initial schedule  $s_0$
- $frag_k^{size}$  is the % of initially non-fixed courses that will be part of the fragment, i.e that will be fixed at their value in the current solution at a given iteration
- The fragment selection heuristic choosing the courses that will be part of the fragment
- $time_{iter}$  is the time limit for an iteration of FAS

**Initial schedule** The initial schedule  $s_0$  is either a user input or an initial schedule built such that it respects all the administrative constraints.

**Fragment selection heuristic** The fragment selection can be random, specific to a problem, generic or a combination. A good heuristic is one that:

- would keep important variables, which are more likely to improve the objective, out of the fragment
- would keep related variables, which more likely allows variable to be assigned differently out of the fragment
- would fix variables when they are equal or near their value in the optimal solution of the initial optimization problem

Since all these aspects are hard to quantify, a random selection is often preferred. Random selection heuristic leads to surprisingly good results in general, has the advantage to be totally generic and provide an excellent diversification. These are the reasons why we choose this selection heuristic.

**Total time** The total time:  $time$  allowed to the algorithm is an input. It can be chosen freely by the user depending on the time he disposes to generate a schedule.

**Maximal time of an iteration** The maximal time limit:  $time_{iter}$  for an iteration of FAS is also a user input. It can be chosen depending on the minimal number of iterations he wants to ensure. The only condition is that  $time_{iter} < time$ . The minimal number of iterations can be computed as:  $\left\lfloor \frac{time}{time_{iter}} \right\rfloor$ .

**Fragment size** Fragment size and maximal iteration time are strongly linked parameters. A rule of thumb is that an iteration of FAS should never be stopped by the time limit:  $time_{iter}$ . If the fragment size or the time limit are too small, neighborhoods are too sparsely explored and consequently FAS might discard promising neighborhoods or miss improving solutions. We could fix the size of the fragment to a constant value for each iteration but then we should have to set a big size to be sure that an iteration would not be stopped by the time limit. We consider here a smarter approach: let the size of the fragment adapt itself based on the output of CPLEX<sup>®</sup> at each iteration. If CPLEX<sup>®</sup> is stopped by the time limit at an iteration then we increase the fragment size. If, on the other hand, CPLEX<sup>®</sup> found the optimal solution then we decrease the fragment size. It follows that the maximal time of an iteration should not be too large or else we would not perform enough iterations but not too small either since we would have too large fragments.

Now we will detail our FAS algorithm using CPLEX<sup>®</sup>, with pseudo-code given by Algorithm 4, solving  $M_{ILP-GD}(E)$ . This algorithm will adjust a number of courses of the fragment in an adaptive way in order to be able to do any iteration within the maximal time of  $time_{iter}$  and improving the final schedule in the sense of  $P^{GD}$ .

In order to control the duration of the steps of the algorithm we introduce timestamps. A timestamp is a value denoting the time at which certain event occurred. The practice of recording timestamps in a consistent manner along with actual data is called timestamping. We actually record when FAS begins and when it has to stop in order to control if we launch new iterations and in if so to supervise the duration of each iteration. *Time.currentMillis()* will help us in this task. This function returns the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 which is the Coordinated Universal Time (CUT).

Let us now introduce the auxiliary function: *iterFAS* that actually performs an iteration of FAS. This function returns a boolean value based on the CPLEX<sup>®</sup> output indicating if the fragment size must be increased (false) or decreased (true) and possibly update the best schedule. *iterFAS* takes 5 arguments:

- *s<sub>input</sub>* is an object representing the best schedule at current iteration
- *frag* the fragment of courses, i.e the set of courses whose corresponding variables in *s* must be fixed at their current time slot in the schedule *s<sub>input</sub>*
- *time<sub>iter</sub>* is the maximum time allowed to CPLEX<sup>®</sup> to solve the program
- *bestObjective* is an object containing  $P^{GD}(s)$
- *endTime* is the timestamp at which FAS has to end

*iterFAS* does the following operations at each iteration, in order:

1. It extracts the partial schedule  $s_{input}(frag)$  from the schedule *s<sub>input</sub>* and builds the  $M_{ILP-GD}(E, s_{input}(frag))$  model of Table 4.1.
2. It solves the model using CPLEX<sup>®</sup> with the maximal time stopping criterion of  $\min \left\{ time_{iter}, \left\lfloor \frac{endTime - Time.currentMillis()}{1000} \right\rfloor \right\}$  seconds, i.e if the total time left for the algorithm is less than the maximal time for an iteration then the we run CPLEX<sup>®</sup> for the total time left. FAS also gives an initial solution as MIP start to CPLEX<sup>®</sup> built from the schedule *s* which once again aims to speed up the resolution.
3. Upon problem resolution or if the maximal time is up, *iterFAS* does the following:
  - (a) If an optimal schedule  $s^*$  is found then it follows that  $P^{GD}(s^*) \leq P^{GD}(s_{input})$ . Consequently:
    - We replace the current schedule *s<sub>input</sub>* by  $s^*$ .
    - We update *bestObjective* with the  $P^{GD}(s^*)$ .
    - We return the boolean true.
  - (b) Else if time is up then let  $s'$  be the feasible schedule obtained
    - i. If the relative MIP gap is less or equal to 5% and  $P^{GD}(s') \leq P^{GD}(s_{input})$  then as result:

- We replace the current schedule  $s_{input}$  by  $s'$ .
  - We update  $bestObjective$  with the  $P^{GD}(s')$ .
  - We return the boolean true.
- ii. Else If  $P^{GD}(s') \leq P^{GD}(s_{input})$  then as result:
- We replace the current schedule  $s_{input}$  by  $s'$ .
  - We update  $bestObjective$  with the  $P^{GD}(s')$ .
  - We return the boolean false.
- iii. In any other case we return the boolean false and do nothing else.

Another mechanism of the FAS algorithm is that it works in two different modes to adapt the size of the fragment. The algorithm begins in a fast start mode in order to obtain rapidly a fragment size such that CPLEX<sup>®</sup> would approximately  $time_{iter}$  seconds to find the optimal solution. We are in this mode when  $fastStart$  is set to true and we exit it the first time  $iterFas$  returns false. In the second mode, when  $fastStart$  is set to false, we still adapt the fragment size but since we make the assumption that we are near to the threshold where CPLEX<sup>®</sup> would approximately  $time_{iter}$  seconds to find the optimal solution we do it in a more smoother way. In the first mode,  $frag_k^{size}$  is decreased by 3 percent at each iteration while in the second mode, it is increased or decreased by 1 percent.

Finally, the iteration mechanism of the FAS algorithm works as follows. While we have at least half the maximal time of an iteration left we make an iteration of FAS calling the auxiliary function:  $iterFAS$  with the fragment,  $frag$  built with  $\left\lfloor \frac{frag_k^{size}}{100} (|E| - |E_{fix}|) \right\rfloor$  courses taken from  $E \setminus E_{fix}$  randomly without replacement. In the special case where the fragment contains no courses,  $iterFAS$  would solve  $M_{ILP-GD}(E)$ . In this special case if it returns true we know that we have the best schedule with a maximum relative MIP gap of 5% and we can stop the FAS algorithm.

To summarize, the main advantages of this method is that the problem size is reduced and it will be possible to obtain good schedules of UCT-EPL-GD(E) faster. The counterpart is that the schedule obtained with this method is often a local minimum of the penalty function  $P^{GD}$ .

### 5.2.3 Third algorithm: Iterated Variable Neighborhood Search

The main reason we chose to test meta-heuristics on UCT-EPL is because it is the most studied solving method nowadays. Most of the papers of the past 20 years that deal with UCT are comparisons between a new heuristic method and previous ones. Good results were often obtained and presented at the PATAT conference [QBML08]. Another advantage is that it can deal with non linear objective functions and constraints.

**Algorithm 4** FAS(*time*: int, *time<sub>iter</sub>*: int, *s<sub>0</sub>*: Schedule): Schedule *s*


---

```

1: function FAS(time: int, timeiter: int, s0: Schedule)
2:   s ← s0
3:   bestObjective ← new Long(Long.MAX_VALUE)
4:   k ← 1
5:   beginTime ← Time.currentMillis()
6:   endTime ← beginTime + 1000 * time
7:   fragSizek ← 99
8:   fastStart ← true
9:   while (endTime − Time.currentMillis()) >  $\frac{time_{iter}}{2} * 1000$  do
10:     fragLengthk =  $\left\lfloor \frac{frag_k^{size}}{100} (|E| - |E_{fix}|) \right\rfloor$ 
11:     frag ← randomArray(fragLengthk)
12:     if iterFAS(frag, s, bestObjective, timeiter, endTime) && fragksize = 0 then
13:       break
14:     else if ¬ iterFAS(frag, s, bestObjective, timeiter, endTime) then
15:       fragSizek ← fragSizek > 98 ? 99 : fragSizek + 1
16:       if fastStart then
17:         fastStart ← false
18:     else
19:       fragSizek ← fastStart ? fragksize − 3 : fragksize − 1
20:       fragSizek ← fragksize < 0 ? 0 : fragSizek
21:     k ← k + 1
22:   return s0

```

---

We chose to implement a method known in the literature as the Iterated Variable Neighborhood Search (IVNS) [RDSB<sup>+</sup>02]. We will first present the pseudo-code of the meta-heuristic with its main philosophy then study the proper algorithm that applies this framework to our problem.

### General algorithm

The Iterated Variable Neighborhood Search, similarly to the Hill-Climber, iteratively checks the neighborhood of a current solution and selects the best feasible neighbor. The difference here is that the definition of the neighborhood probabilistically varies from an iteration to another and the local search can restart when the solution does not improve for a certain amount of iterations. The pseudo-code is given below in Algorithm 5:

---

**Algorithm 5** IVNS(*time* : int, *Prob* : list) : Solution *x*


---

```

1: function IVNS(time : int, Prob : list)
2:    $x^* \leftarrow \text{getInitialSolution}()$ 
3:    $x' \leftarrow x^*$ 
4:   blocked  $\leftarrow \text{false}$ 
5:   while runningTime < time do
6:      $i \leftarrow \text{selectNeighborhood}(\text{Prob})$ 
7:      $\tilde{x} \leftarrow \min\{f(x) \mid x \in N_i(x') \subseteq X\}$ 
8:     if  $f(\tilde{x}) \leq f(x')$  then
9:        $x' \leftarrow \tilde{x}$ 
10:    Update blocked
11:    if blocked then
12:      if  $f(x') < f(x^*)$  then
13:         $x^* \leftarrow x'$ 
14:       $x' \leftarrow \text{disturb}(x^*)$ 
15:  Return  $x^*$ 

```

---

The algorithm takes two parameters as input: a running time *time* and a probability distribution *Prob*. It initializes itself by generating an initial solution  $x_0$ . Then it sets the global best solution  $x^*$  to  $x_0$  and the locally best solution  $x'$  to  $x_0$  as well. While a chosen blocking criterion is not met, the algorithm iteratively improves  $x'$  by selecting a neighborhood definition at each iteration according to *Prob* and looking for the best neighbor. Once the algorithm is blocked, it updates the global best solution if necessary and checks for a new starting point for the local search by slightly modifying  $x^*$  using a *disturb()* function.

The IVNS explores a large subset of the solution space by using multiple neighborhoods and avoids being stuck on a local solution. One flaw is that due to the randomness of the search it does not guarantee to find a solution close to the optimum, as in all meta-heuristics.

### Problem specific algorithm

We designed an IVNS algorithm meant to solve  $M_{INP-BSR}$ . The IVNS applied to our problem, with pseudo-code given by Algorithm 6 has the same architecture as the general algorithm.

---

#### Algorithm 6 IVNS

---

```

1: function IVNSUCT-EPL(time: int, blocked1: boolean, blocked2: boolean, ratio: dou-
   ble, (s0: Schedule))
2:    $s^* \leftarrow \text{getInitialSol}()$  OR  $s^* \leftarrow s_0$ 
3:    $s' \leftarrow s^*$ 
4:    $b_1 \leftarrow 0$ 
5:    $b_2 \leftarrow 0$ 
6:   while runningTime < time do
7:      $i \leftarrow \text{selectMove}()$ 
8:      $\tilde{s} \leftarrow \text{applyMethod}(i, s')$ 
9:     if  $P^{BSR}(\tilde{s}) \leq P^{BSR}(s')$  then
10:       $s' \leftarrow \tilde{s}$ 
11:       $b_1 \leftarrow 0$ 
12:     else
13:       $b_1 \leftarrow b_1 + 1$ 
14:      if  $b_1 = \text{blocked1}$  then
15:        if  $P^{BSR}(s') < P^{BSR}(s^*)$  then
16:           $b_2 \leftarrow 0$ 
17:        else
18:           $b_2 \leftarrow b_2 + 1$ 
19:        if  $b_2 = \text{blocked2}$  then
20:           $s' \leftarrow \text{disturb}(s^*, \alpha)$ 
21:        else
22:           $s' \leftarrow \text{disturb}(s', \alpha)$ 
   Return  $s^*$ 

```

---

As observed we need to define 4 sub methods in order to have a complete pseudo-code :  $\text{getInitialSol}()$ ,  $\text{selectMove}()$ ,  $\text{applyMove}(i, s)$  and  $\text{disturb}(\alpha, s)$ . These sub methods are described in small texts below.

- **getInitialSol()**

This method generates a greedy initial schedule. It first assigns all the examinations with fixed time slots. Then it uses a method  $\text{rankedExaminations}()$  that returns the list  $L$  of the non fixed examinations ranked in a decreasing order according to number of conflicts they create  $\bar{C}_i = \sum_{j=1}^n C_{ij}$ . Since it is easy to move a course that does not create a lot of conflicts, it adds successively each element in list  $L$  to the schedule and assigns it to the time slot that minimizes the objective function of the current schedule. It also checks that the time slot assigned produce a feasible a schedule. The algorithm can take an initial solution as input and do not need to use this function if it is the case.

- **selectMove()**

In our case, the neighborhoods are defined by the set of schedules obtained by applying certain move we on the current schedule, such as switching the time slots of two examinations. The method *selectMove()* returns the move to use at each iteration. We chose to use a uniform distribution for picking one of the three moves. So each move has a probability  $1/3$  to be chosen at each iteration.

- **applyMove(i, s)**

This method applies move  $i$  to schedule  $s$ . The three possible moves are the following:

1. select a random course and swap its time slot with the time slot of the course that decreases the most the objective function.
2. select a random course and set its time slot as the one that that decreases the most the objective function.
3. select 4 courses and take the permutation of the respective time slots that decreases the most the objective function.

- **disturb( $\alpha$ , s)**

This method simply selects a list of  $\lceil \alpha n_E \rceil$  random courses ( $n_E$  being the total number of courses) and attributes them a random feasible time slot. One may argue that it could have been a good idea to select the time slots in a greedy approach that minimize the objective at each iteration. However, selecting random time slots allows to diversify the search when a greedy approach would intensify the search in the local neighborhood.

During the local search, if the algorithm did not find any better solution during *blocked1* iterations, it will do the following :

1. it checks if the current schedule is better than the global best one and :
  - if it is the case,  $b_2$  is reset to 0
  - else it will increase  $b_2$  by one unit
2. if  $b_2$  is equal to *blocked2*, it restarts on a new iterate obtained by disturbing the global best solution
3. else it starts on the disturbed current solution

We arbitrarily set *blocked2* to 5 in our case. So our IVNS takes 2 parameters which are *alpha*, further referred as the *ratio*, and *blocked1*, further referred as the *blocked* parameter.

## 5.3 Post-optimization

As it is explained in the problem statement, the inscription of a student to a major (resp. minor) examination can be switched to a minor (resp. major) inscription if it improves  $P^{BSR}$ . We can also consider to switch two students taking the same oral examination in different duplicates. The post-optimization is a heuristic algorithm that minimizes  $P^{BSR}$  by modifying students enrollments as described above. In the first step, it runs through all the students and checks if it is better to switch their major/minor inscriptions. Then it will randomly picks two students following the same oral examination but different duplicates and checks if an enrollments switch is advantageous.

## 5.4 Solver

From the pre-processing, the post-processing and the 3 main optimization methods, it is possible to define a series of solvers, that we will be able to compare in the next chapter.

A solver is defined by first applying the pre-processing. When this is achieved, it applies the FAS algorithm for  $x$  % of the given time. The solutions obtained are used as an initial solution for the CILP algorithm running for  $y$  % of the given time. Then we use the IVNS algorithm to affine the solution obtained for  $z$  % of the time limit, after which the post-processing algorithm is performed. A graphical representation of a solver is given on Figure 5.1.

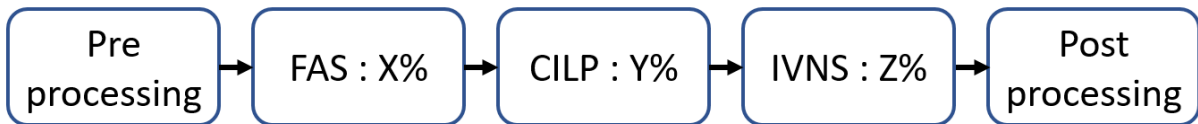


Figure 5.1: Solver architecture

This solver architecture has been chosen for two main reasons. First, since we are trying to find a schedule minimizing  $P^{BSR}$ , the IVNS algorithm must come after FAS and CILP, which are minimizing  $P^{GD}$ . These last two algorithms provide strong solutions due to the efficiency of CPLEX<sup>®</sup> and are then send to the IVNS for refining. On the other hand the resolution of  $M_{LIN-GD}(E)$  using CILP is fasten if a good initial solution is provided. Since FAS produces good solutions in a short amount of time (as it will be observed in the tests chapter), a solver of this form seems to be an efficient choice.

We arbitrarily chose 25 different proportions of  $x$ ,  $y$  and  $z$ , leading to 25 solvers that will be tested. The proportions were chosen in a way that we would end up with the most diversified set of solvers. We wanted to test the three algorithms on their own,

mix the three of them with different proportions and also show the impact of assigning a small percentage of the time to the IVNS. Solvers and their ID numbers are given in Table 5.1

| Solvers ID | % FAS | % CILP | % IVNS | Solvers ID | % FAS | % CILP | % IVNS |
|------------|-------|--------|--------|------------|-------|--------|--------|
| 1          | 0     | 0      | 100    | 13         | 25    | 0      | 75     |
| 2          | 0     | 100    | 0      | 14         | 25    | 25     | 50     |
| 3          | 0     | 25     | 75     | 15         | 25    | 50     | 25     |
| 4          | 0     | 50     | 50     | 16         | 25    | 75     | 0      |
| 5          | 0     | 75     | 25     | 17         | 47    | 47     | 6      |
| 6          | 0     | 94     | 6      | 18         | 50    | 0      | 50     |
| 7          | 100   | 0      | 0      | 19         | 50    | 25     | 25     |
| 8          | 10    | 15     | 75     | 20         | 50    | 50     | 0      |
| 9          | 10    | 40     | 50     | 21         | 5     | 0      | 95     |
| 10         | 10    | 65     | 25     | 22         | 72    | 22     | 6      |
| 11         | 10    | 84     | 6      | 23         | 75    | 0      | 25     |
| 12         | 22    | 72     | 6      | 24         | 75    | 25     | 0      |
|            |       |        |        | 25         | 94    | 0      | 6      |

Table 5.1: Solvers

The parameters for a given solver are the  $time_{iter}$  of the FAS algorithm and the  $blocked$  and  $ratio$  for the IVNS.

## 5.5 Summary

Now that we have defined the complete solver and provided the full description of its algorithms we can run numerous tests on it. The chapter that follows determines which of the  $\{time_{iter}, blocked, ratio\}$  combinations are the best for the 25  $x/y/z$  proportions defined for several time windows and then selects a default time window and the best solver for this one.

# TESTS

---

This chapter contains all the tests that have been conducted. They are separated into three main categories that are shortly introduced below.

## Parameters tuning & time setting

As explained in the previous chapter, two out of the three main optimization algorithms have parameters that can be adjusted. The goal here is to find the best parameters configuration for each solver. Since we do not have any constraining time limit, we will run the different solvers for increasing reasonable time windows chosen arbitrarily. Once we reach a time window where there is no significant improvement in the performances, we will set this time window as default.

## Solvers comparison

Once the best parameters and the default time have been set, we compare the solvers between each other to determine which one is the best among them.

## Complementary tests

In this last test batch we will explore additional features that might be interesting for EPL. While the two preceding tests have been designed to produce a solver that responds to the initial demand, we test here some modifications to the administrative specifications or the solver process that may lead to better results.

Tests have been implemented in Java 8 because we observed empirical good performances with the CPLEX<sup>®</sup> Java API. Due to their large number, these tests ran on the CISM clusters of UCL, and more precisely using the Manneback Cluster. The tests ran on an AMD CPU named Bulldozer Opteron 6276. It has a frequency of 2.3 GHz, contains 16 cores and has 16 MB of memory cache. We imposed a limit of 4 cores in order to use the efficiency of CPLEX<sup>®</sup> multi-threading <sup>1</sup>, each using at most 8 MB of

---

<sup>1</sup>See IBM support

memory cache. This corresponds to 32 MB of memory, which is the standard Random Access Memory for a good desktop computer.

As explained in Chapter 3,  $P^{BSR}(s)$  serves as a standard penalty function for any schedule. Therefore this function is used in this chapter as quality metric to evaluate every schedule and to compare the schedules between each other.

The large amount of tests do not allow to report the full numerical results in the Thesis. They can be found in a repository via the the following URL :

<https://github.com/adbrogniet/UCT-EPL-tests>

## 6.1 Data sets

Before presenting the tests, we provide a description of the data sets used and the methodology chosen to adapt them to the specifications.

### 6.1.1 Methodology

We received from Mr. Glineur, Vice dean at EPL and supervisor of this Thesis, 12 data sets regrouping examination sessions for the academic years that begin in 2014, 2015 and 2016 and 2017. We have inscriptions for EPL courses and additionally courses from other faculties, denoted as ex-EPL courses, for the years 2014 and 2017. For each year, we have the 3 sessions of January, June and August, that we will respectively denoted by Q1, Q2 and Q3. For each session of 2014 and 2017 we generate 2 sets. The first one where only the ex-EPL courses are given with their time slot fixed beforehand. Then a second one with, in addition to the ex-EPL courses, the bachelor courses fixed as the person in a way that minimizes the unwanted gaps. Since for the years 2015 and 2016 we do not have ex-EPL courses inscriptions, we generate again 2 sets per session but no courses are fixed in the first one while in the second, only the bachelor courses are fixed.

However, data sets were not complete regarding the specifications given in Chapter 3 and we had to manually perform some improvements on the data sets. We were missing the following information :

#### Examination formats

Since the data sets did not specify whether an examination is oral or written, we had to manually set this characteristic. To perform this, information was gathered from the UCL website. As the information was sometimes missing, we do not guarantee that the oral courses in the data sets perfectly correspond to reality. Nevertheless, we can

assume that the data sets are quite representative of the true cases and that the solver behavior will not be impacted too much by this approximation.

### **Major/minor examinations**

The information on which minor/major curriculum a course belongs is clearly available on the UCL website, so it was directly added to the data sets.

### **Unavailabilities**

As this information is a new feature added to the current ways examination sessions are created at EPL, this information does not exist at all. So we randomly created some unavailabilities, at a reasonable level, so that it does not alter the solver behavior.

### **Student major and minor options**

The major and minor options of each student was not given with the initial data. Since we knew their courses enrollments and whether a course belongs to a major or minor option, we were able to recover this information. As some courses are present in multiple options, we do not guarantee that the computed information are entirely exact, but regarding the large number of students, we can assume it is strongly representative of a real case.

### **Fixed examinations**

The information on examinations schedules from other faculties was not available. Therefore we set this input in random but realistic way, such that there are no conflicts between two examinations from the same faculty.

## **6.1.2 Description**

The sets are listed and indexed for later reference in the Tables 6.1,6.2,6.3 and 6.4. Note that the features are defined by the acronyms EF and BF. They respectively denote the cases where ex-EPL courses are available and bachelor courses are fixed.

| Data sets ID               | 1       | 2       | 3       | 4       | 5       | 6       |
|----------------------------|---------|---------|---------|---------|---------|---------|
| Session                    | 2014 Q1 | 2014 Q1 | 2014 Q2 | 2014 Q2 | 2014 Q3 | 2014 Q3 |
| Features                   | EF      | EF+BF   | EF      | EF+BF   | EF      | EF +BF  |
| # Courses                  | 223     | 223     | 289     | 289     | 256     | 256     |
| # Students                 | 1741    | 1741    | 1740    | 1740    | 986     | 986     |
| # Time slots               | 38      | 38      | 48      | 48      | 36      | 36      |
| # Fixed courses            | 89      | 100     | 57      | 69      | 38      | 59      |
| # Major/Minor examinations | 16      | 16      | 27      | 27      | 0       | 0       |
| # Oral examinations        | 13      | 13      | 12      | 12      | 3       | 3       |
| # Unavailabilities         | 3       | 3       | 12      | 12      | 4       | 4       |

Table 6.1: Data sets 1-6

| Data sets ID               | 7       | 8       | 9       | 10      | 11      | 12      |
|----------------------------|---------|---------|---------|---------|---------|---------|
| Session                    | 2015 Q1 | 2015 Q1 | 2015 Q2 | 2015 Q2 | 2015 Q3 | 2015 Q3 |
| Features                   |         | BF      |         | BF      |         | BF      |
| # Courses                  | 173     | 173     | 181     | 181     | 269     | 269     |
| # Students                 | 2116    | 2116    | 1976    | 1976    | 1290    | 1290    |
| # Time slots               | 32      | 32      | 46      | 46      | 32      | 32      |
| # Fixed courses            | 0       | 22      | 0       | 19      | 0       | 27      |
| # Major/Minor examinations | 25      | 25      | 26      | 26      | 0       | 0       |
| # Oral examinations        | 9       | 9       | 9       | 9       | 8       | 8       |
| # Unavailabilities         | 4       | 4       | 5       | 5       | 3       | 3       |

Table 6.2: Data sets 7-12

| Data sets ID               | 13      | 14      | 15      | 16      | 17      | 18      |
|----------------------------|---------|---------|---------|---------|---------|---------|
| Session                    | 2016 Q1 | 2016 Q1 | 2016 Q2 | 2016 Q2 | 2016 Q3 | 2016 Q3 |
| Features                   |         | BF      |         | BF      |         | BF      |
| # Courses                  | 146     | 146     | 164     | 164     | 262     | 262     |
| # Students                 | 2258    | 2258    | 2156    | 2156    | 1453    | 1453    |
| # Time slots               | 38      | 38      | 46      | 46      | 36      | 36      |
| # Fixed courses            | 0       | 19      | 0       | 13      | 0       | 25      |
| # Major/Minor examinations | 15      | 15      | 29      | 29      | 0       | 0       |
| # Oral examinations        | 6       | 6       | 7       | 7       | 8       | 8       |
| # Unavailabilities         | 5       | 5       | 2       | 2       | 4       | 4       |

Table 6.3: Data sets 13-18

| Data sets ID               | 19      | 20      | 21      | 22      | 23      | 24      |
|----------------------------|---------|---------|---------|---------|---------|---------|
| Session                    | 2017 Q1 | 2017 Q1 | 2017 Q2 | 2017 Q2 | 2017 Q3 | 2017 Q3 |
| Features                   | EF      | EF + BF | EF      | EF + BF | EF      | EF + BF |
| # Courses                  | 223     | 223     | 267     | 267     | 268     | 268     |
| # Students                 | 1990    | 1990    | 1974    | 1974    | 1203    | 1203    |
| # Time slots               | 38      | 38      | 44      | 44      | 36      | 36      |
| # Fixed courses            | 78      | 89      | 85      | 97      | 75      |         |
| # Major/Minor examinations | 15      | 15      | 27      | 27      | 0       | 0       |
| # Oral examinations        | 6       | 6       | 6       | 6       | 3       | 3       |
| # Unavailabilities         | 5       | 5       | 6       | 6       | 0       | 0       |

Table 6.4: Data sets 19-24

## 6.2 Parameters tuning

### 6.2.1 Methodology

For given training data sets, we will study for each of the 25 solvers and for increasing time limit, which  $\{time_{iter}, blocked, ratio\}$  combination provides the best examinations schedule. Since each solver is stochastic, multiple runs should be performed for each test configuration and a statistical measure should be defined. To compute the best parameters of a solver, we will use a mean-variance approach based on  $P^{BSR}$ . This approach consists of measuring the quality of a parameters combination as the mean of the sample of  $P^{BSR}$  obtained from the multiple runs. The variance will account for a measure of dispersion of the sample. The criterion for a combination to be chosen as the best one is that his mean is the lowest, provided that his variance is not excessive compared to other combinations.

The two most popular measures when comparing stochastic algorithms are usually the mean and the median as a measure of quality, the variance and the interquartile range as a measure of dispersion. We choose to go for the mean for one reason. Since we are running a large amount of tests, we do not have the opportunity to perform numerous runs. Therefore, the mean seems to be the right fit, since if a run results in a very bad solution, it will be reflected in the mean.

On the other hand, the median-IQR approach provides good visual support, via the use of box plots. For this reason we will refer to box plots for determining if a solver is not relevant or if the pre-processing is efficient, as explained in the following paragraph.

Besides the main willingness to compute the best parameters configurations, these tests purchase three other subsidiary goals :

1. **Validating the efficiency of the pre-processing**

We replicated the first round of tests without pre-processing, where students are randomly assigned to an oral examination slot.

2. **Deleting irrelevant solvers**

While the rigorous comparison between solver performances is performed in the second batch of tests, it is already possible using box plots to assess whether a solver is completely irrelevant or not.

3. **Determining the default time**

Each round of tests correspond to a time limit. Whenever a time is reached for which the solvers do not progress anymore in terms of  $P^{BSR}$  this time window will be selected.

We will report here considerations on the three goals described above and some additional remarks on the solvers behaviors, rather than simply enunciate the best set of parameters for each solver.

### 6.2.2 Test configuration

**Time configuration** We choose the following reasonable time windows arbitrarily: 30 min - 1h - 3h

**Data sets used** We used the following diversified sets (see Subsection 6.1 to recall the features of each set): 2 - 9 - 10

#### Parameters

- $time_{iter}$  : 2% - 5% - 10% - 20% of the FAS running time
- $blocked$  : 10 - 20 - 30
- $ratio$  : 5% - 10% - 20%

**Runs** We run every test 5 times.

### 6.2.3 Results

#### 6.2.3.1 Round 1 - 30 min

Figures 6.1, 6.2 and 6.3 depict the box plots obtained in terms of  $P^{BSR}$  for each of the 25 solvers, at their best parameters configuration.

The first major observation that we can make is that solvers 1 to 6 seem to be less efficient than the other ones. These solvers correspond to the case where 0 % of the running time is assigned to the FAS algorithm. This is not surprising since this algorithm provides good solutions quite fast, contrarily to the CILP and IVNS algorithm. Indeed solvers 1 and 2 correspond respectively to the 100 % IVNS solver and 100 % CILP solver. We observe that the solutions obtained are quite bad, while solver 7, corresponding to 100 % FAS, provides one of the best results. From this consideration, we could expect to consider only solvers applying FAS in the future. However, we will still consider solvers 1 to 6 in the second round of 1 hour, to prevent from early considerations.

Another consideration that should be noted is the fact that the less examinations are fixed, the better are the solutions. The most constrained data set is set 2, containing fixed ex-EPL courses, and all objectives lie within 45000 and 75000. Then comes data set 10, with only bachelor courses fixed, with objectives contained between 20000 and 40000. For data set 9 with no fixed courses, objectives are between 10000 and 40000

(except for solver 6). This result was expected to happen, as fixed courses sometimes impose short times between examinations.

In addition to the loss in term of overall quality, fixing courses also seems to increase the dispersion of the solution quality. As observed on Figures 6.1, 6.2 and 6.3, the Interquartile Ranges increases as a data set has more fixed courses.

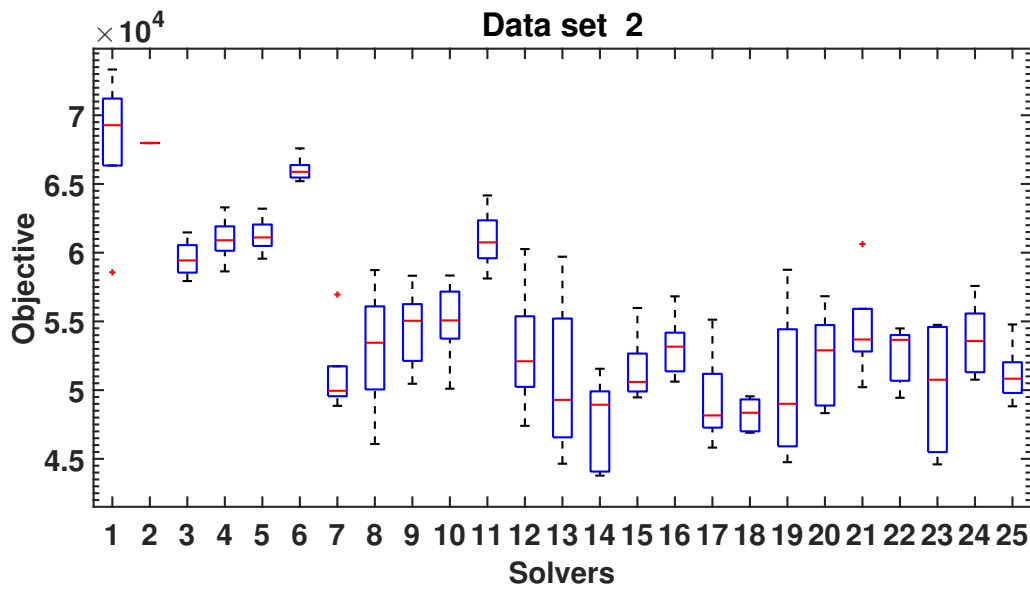


Figure 6.1: Parameters tuning - 30 min - Data set 2: solvers with their best configurations

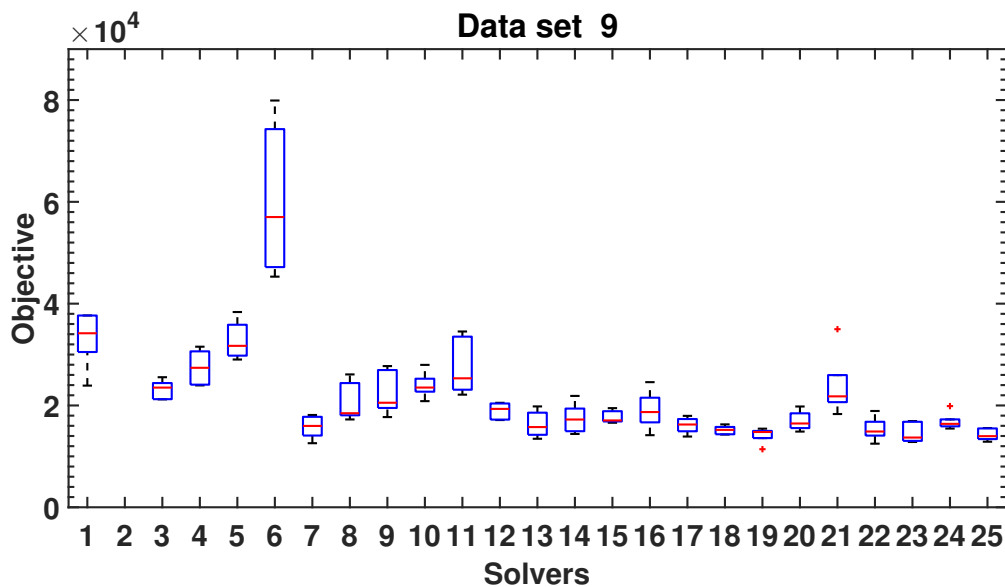


Figure 6.2: Parameters tuning - 30 min - Data set 9: solvers with their best configurations

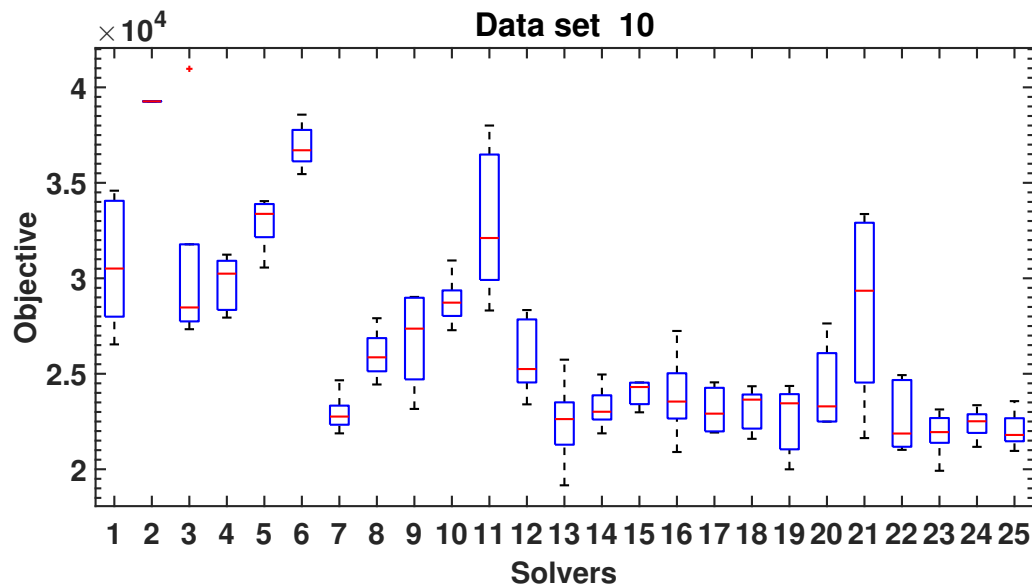


Figure 6.3: Parameters tuning - 30 min - Data set 10: solvers with their best configurations

As stated before, we decide to apply this round of tests for identical solvers and parameters combination but where the pre-processing step has been replaced by a random assignments of student in oral examinations. All the solutions from this round have been compared to all the solutions of the same round but with the random assignment instead of the pre-processing on Figures 6.4, 6.5 and 6.6. These Figures show the undeniable benefit of the pre-processing step, as no solutions generated using the regular solvers are worst than the ones from random assignment (except two outliers in data set 9).

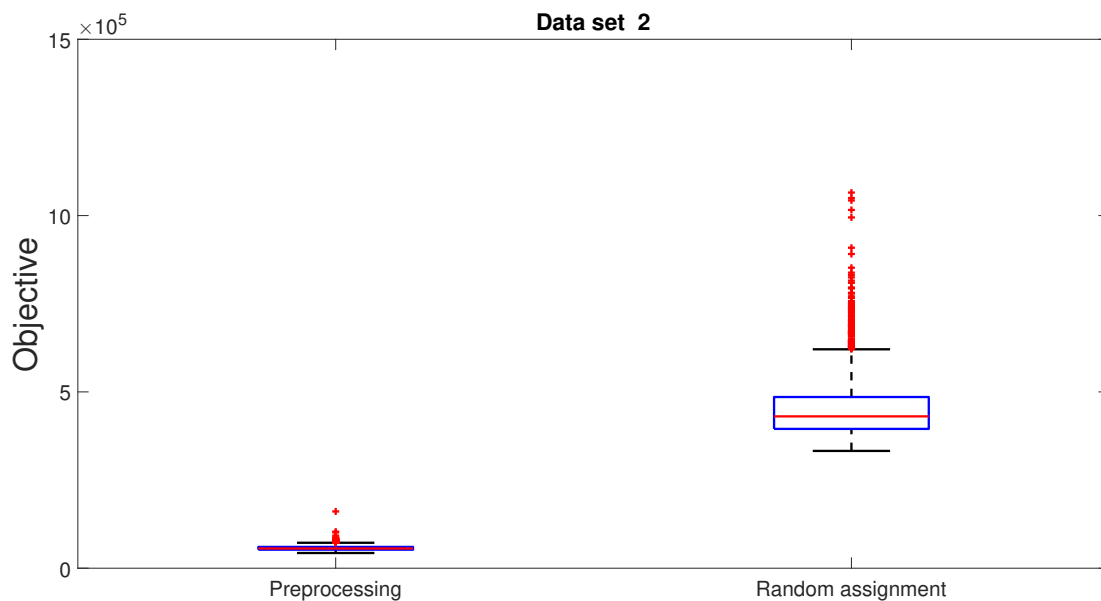


Figure 6.4: Parameters tuning - 30 min - Data set 2: pre-processing performance

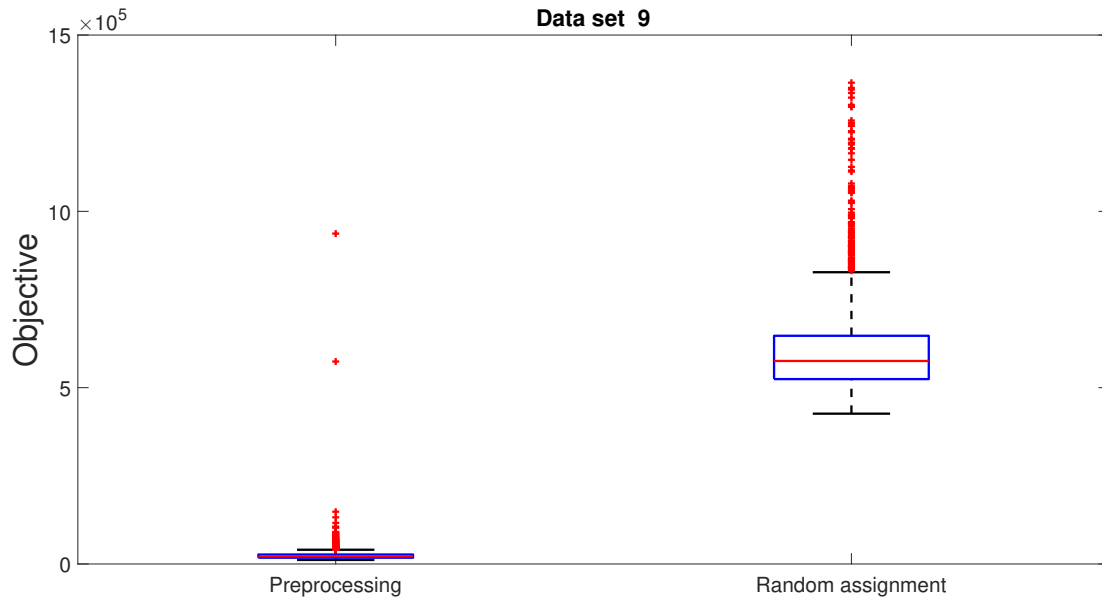


Figure 6.5: Parameters tuning - 30 min - Data set 9: pre-processing performance

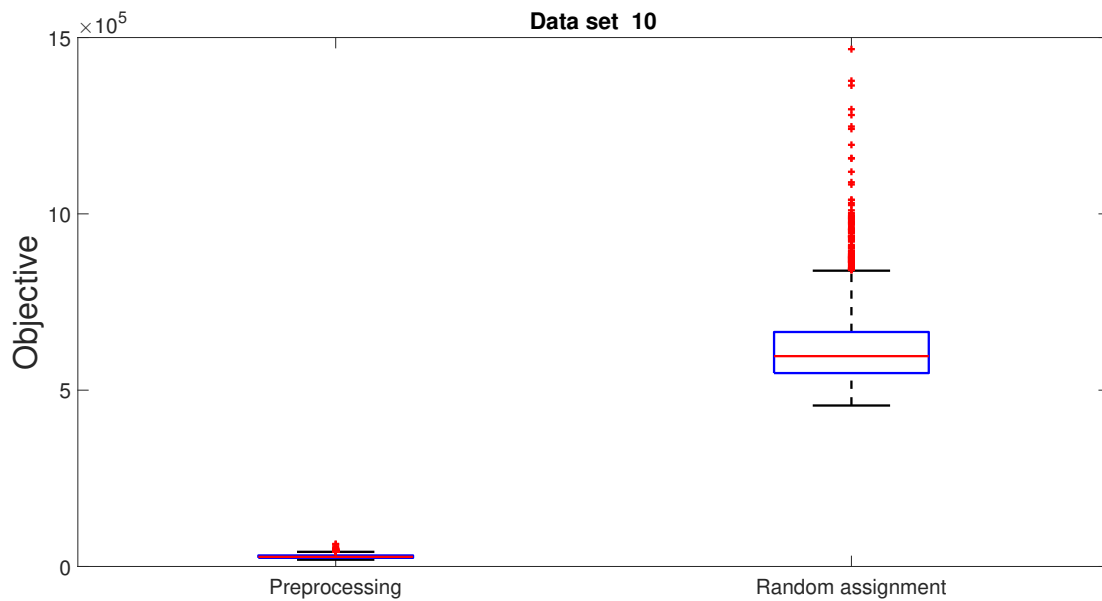


Figure 6.6: Parameters tuning - 30 min - Data set 10: pre-processing performance

### 6.2.3.2 Round 2 - 1h

Results for this round are depicted on Figures 6.7, 6.8 and 6.9.

As for the precedent round, solvers 1 to 6 are way worst than the others. For the reason enunciated in the previous section, we decide to discard these solvers, as they will not provide efficient solutions in comparison with solvers 7 to 25.

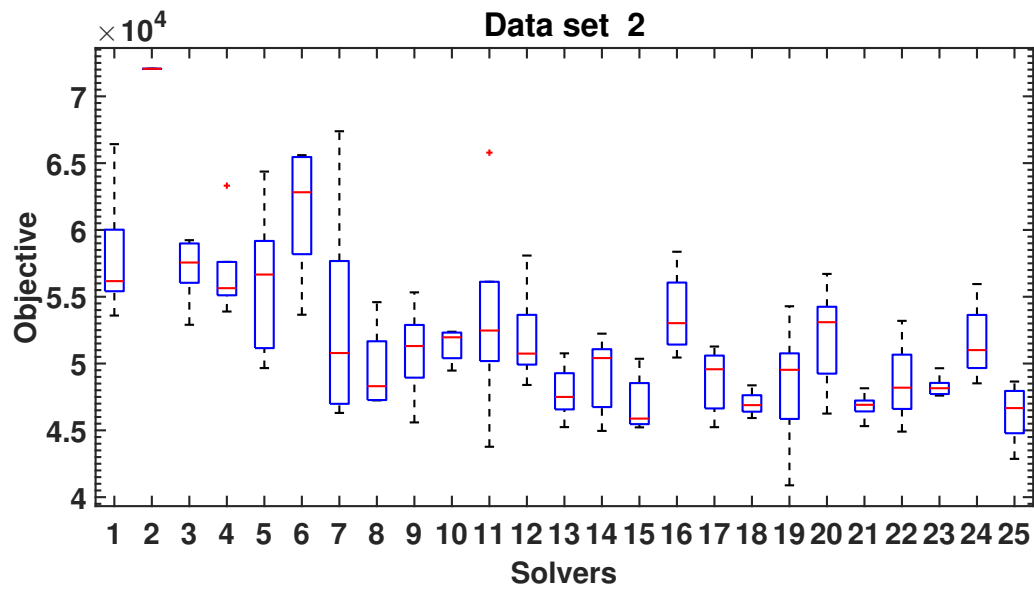


Figure 6.7: Parameters tuning - 1 hour - Data set 2: solvers with their best configurations

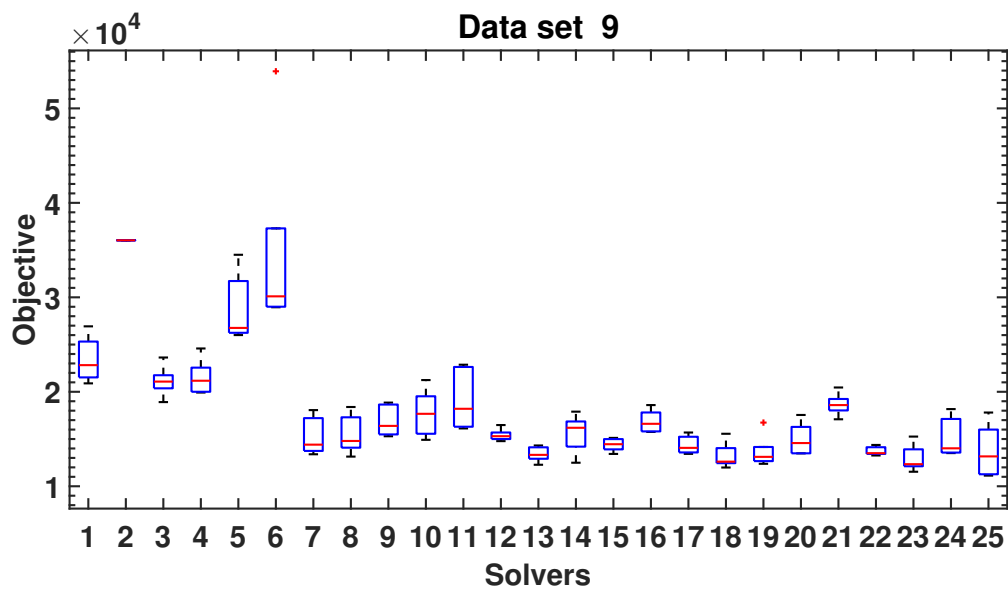


Figure 6.8: Parameters tuning - 1 hour - Data set 9: solvers with their best configurations

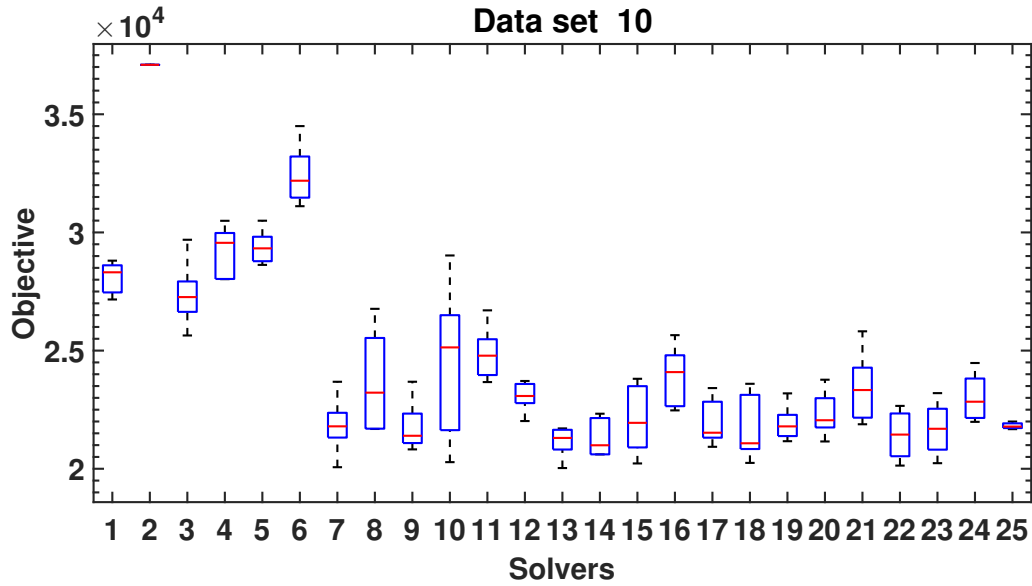


Figure 6.9: Parameters tuning - 1 hour - Data set 10: solvers with their best configurations

### 6.2.3.3 Round 3 - 3h

The box plots for this round are given on Figures 6.10, 6.11 and 6.12. While they do not allow us to distinguish a solver that outperforms the other, one other interesting observation can be made. Since the CILP and the FAS are not minimizing  $P^{BSR}$ , we observe that it is always more efficient to refine the schedules with a small duration of IVNS. Indeed, even if the solvers not using the IVNS algorithm, i.e. solvers 7, 16, 20 and 24, are not necessarily the worst, they are never the best ones and often lead to bad schedules. For this reason we will not consider these 4 solvers anymore for the solvers comparison.

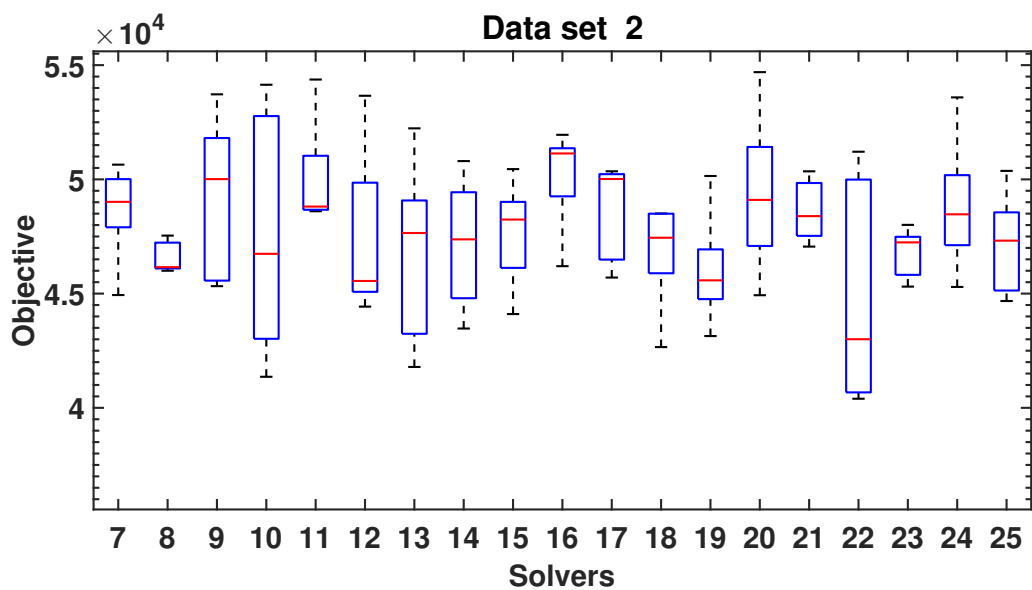


Figure 6.10: Parameters tuning - 3 hours - Data set 2: solvers with their best configurations

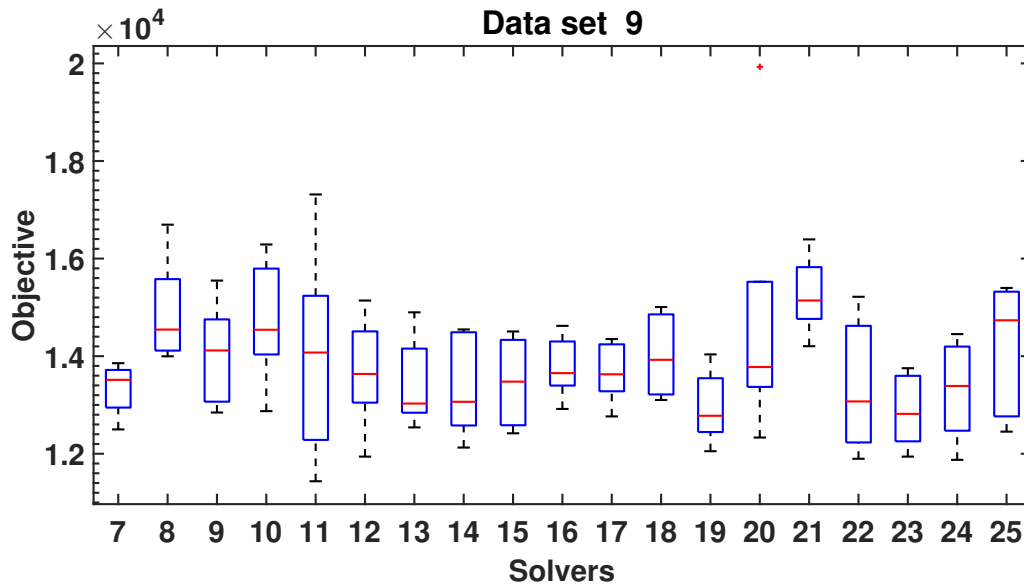


Figure 6.11: Parameters tuning - 3 hours - Data set 9: solvers with their best configurations

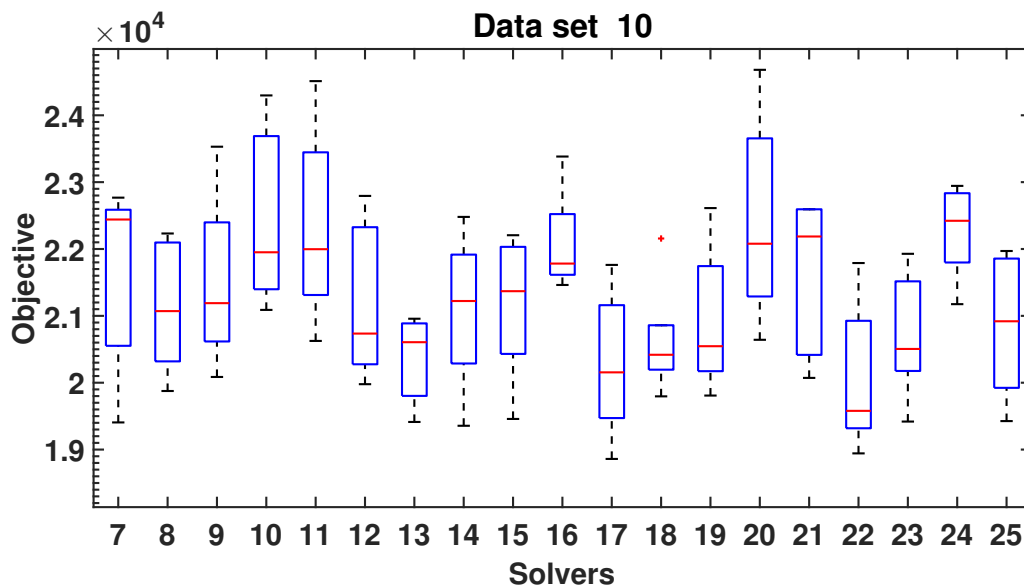


Figure 6.12: Parameters tuning - 3 hours - Data set 10: solvers with their best configurations

We see that increasing the time window has not led to major decreases of objective function with respect to the previous run. In fact we can observe that when running the solvers for 12h, objectives are practically identical. Therefore we can consider 3 hours as the default running time of the solver.

## 6.3 Solvers comparison

### 6.3.1 Methodology

This batch of tests compares the performances of the 15 remaining solvers for the standard time window, which is 3 hours, given their best parameters combination. Ideally, we want to distinguish which solver is the most robust and which one leads to better objective function values on average.

Comparison between solvers will be assessed via the use of performance profiles. Performance profiles, introduced in 2001 by Elizabeth D. Dolan and Jorge J. Moré [DM01b], are more and more popular when it comes to benchmarking optimization. This concept is introduced in the following subsection.

#### 6.3.1.1 Performance profiles

Performance profiles answer the following question:

*Given a set of solvers and a set of data sets, what is the probability that a particular solver will win over the other solvers on the basis of a performance measure ?*

To achieve this, we define the following:

- a set of solvers  $\mathcal{S}$
- a set of data  $\mathcal{D}$
- a performance measure  $m : \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{R}$  such that  $m_{s,d}$  gives the performance of solver  $s$  on data set  $d$ .

From these we can deduce the performance ratios:

$$r_{s,d} = \frac{m_{s,d}}{\min_{s \in \mathcal{S}} m_{s,d}}$$

which gives the performance of a solver compared to the best solver for a given data set. It is then possible to define the performance profile of a solver  $s \in \mathcal{S}$  denoted by  $\rho_s$  as the cumulative distribution function of the performance according to a ratio :

$$\rho_s : [1, \tau_{max}] \rightarrow [0, 1] : \rho_s(\tau) \rightarrow \frac{1}{|\mathcal{D}|} |\{d \in \mathcal{D} \text{ s.t. } r_{s,d} \leq \tau\}|$$

with  $\tau_{max} = \max_{s \in \mathcal{S}, d \in \mathcal{D}} r_{s,d}$ . So  $\rho_s(\tau)$  is the probability for solver  $s$  that a performance ratio  $r_{s,d}$  is within a factor  $\tau$  of the best possible ratio. For any given ratio  $\tau$ , the highest the  $\rho_s(\tau)$  is, the better the solver is. The probability  $\rho_s(1)$  is the probability that solver  $s$  outperforms all the other solvers.  $\rho_s(\tau)$  for large values of  $\tau$  is the probability that  $s$

wins given that we tolerate the solution to be within a factor  $\tau$  of the best solution. This might be used to assess the solver robustness.

Note that performance profiles are defined for deterministic algorithms, as the measure  $m_{s,d}$  is supposed to be unique. Since we are interested here in comparing stochastic solvers, we need to adapt performance profiles to this extent. Since we considered the mean as our selection criterion in the first batch, we do the same here and consider  $m_{s,d}$  as the mean of the sample of  $P^{BSR}$  values obtain using solver  $s$  on data set  $d$ . The criterion to chose a solver among all the others is to be the one with the highest  $\rho_s(1)$ , as we are interested to the probability to provide the best schedule.

### 6.3.2 Test configuration

**Time configuration** The time duration is the default time: 3 hours

**Data sets used** We used the 24 data sets available

**Parameters** Parameters for each solver have been set to their best configuration, according to the results of the previous batch.

**Runs** We run every test 20 times.

### 6.3.3 Results

The performance profiles of the five best solvers, i.e. the ones with the largest  $\rho_s(1)$ , are given in Figure 6.13

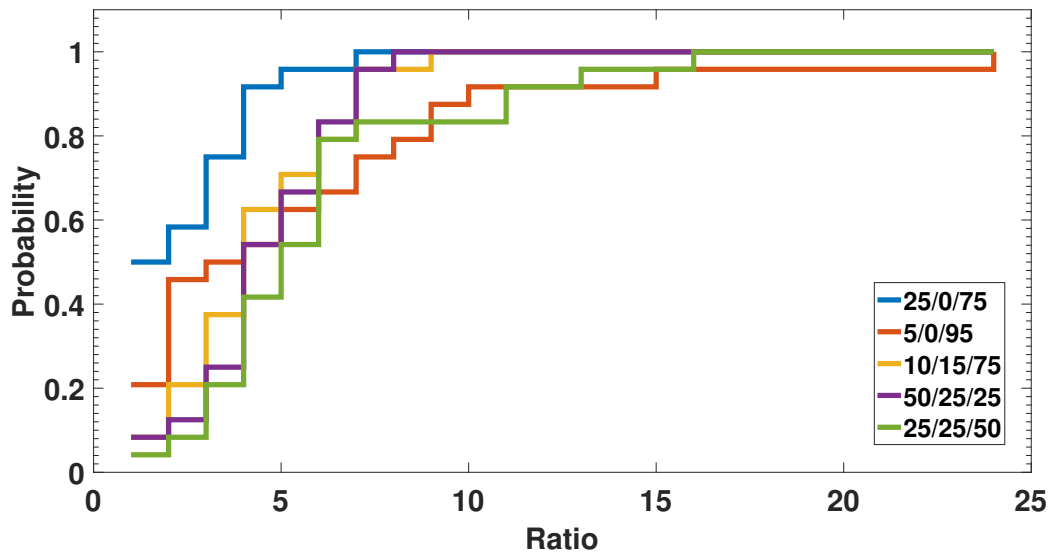


Figure 6.13: Performances profiles of the five best solvers

We observe that solver 25/0/75 has 50 % chance to provide the best solution among the 15 solvers. It is also the most robust as it is always the one with the highest probability to be within any given ratio. The optimal solver is then solver 25/0/75 running with  $time_{iter} = 2\%$ ,  $blocked = 30$  and  $ratio = 5\%$ .

## 6.4 Complementary tests

This section presents additional tests performed after we computed the optimal solver. We want here to achieve the three following goals :

1. Illustrate why using CPLEX<sup>®</sup> for solving  $M_{ILP-GD}$ , as in CILP, is not efficient
2. Check if realistic modifications on the administrative constraints may improve the quality of the examinations schedule:
  - We investigate whether scheduling only one examination instead of two for major/minor courses brings a real benefit
  - We observe the evolution of  $P^{BSR}$  when we add/suppress days to/of the examination session
3. Test to observe a decrease in  $P^{GD}$  when we use the nested decomposition on  $M_{ILP-GD}$  for a given time window.

### 6.4.1 Inefficiency of CPLEX<sup>®</sup> on $M_{ILP-GD}(\tilde{E})$

#### 6.4.1.1 Methodology

As specified multiple times in the report, the Branch and Bound tree is too dense for regular instances of UCT-EPL, so that CPLEX<sup>®</sup> is not able to provide any viable solution of  $M_{ILP-GD}$ , even for large time limits. We ran CPLEX<sup>®</sup> for 24 hours on  $M_{ILP-GD}(Cl)$  where  $Cl$  is the set of the courses belonging to the maximal clique in the conflict graph of data set 9. We showed that even for this sub-instance, CPLEX<sup>®</sup> cannot obtain a solution with a reasonable (5 %) relative MIP gap.

#### 6.4.1.2 Results

At the end of the running time, the Branch and Bound algorithm explored a solution tree with a size of 35 GB without converging. In fact, the best solution found in terms of  $P^{GD}$  has a relative MIP gap of 57.36 %. This shows why using CPLEX<sup>®</sup> on UCT-EPL-GD has no guarantee of optimality and produces suboptimal solutions. Actually, providing a good schedule for a number of courses equivalent to this clique size, which is 12, might be done manually while in this case CPLEX<sup>®</sup> does not return an optimal schedule.

## 6.4.2 Suppressing minor/major duplicates

### 6.4.2.1 Methodology

We performed on the data sets of the January and June sessions the same tests as in Section 6.3, except that we considered major/minor examinations as normal ones that can only be scheduled once. Note that since students inscriptions from outside EPL have not been taken into account, this may lead to conflict situations in real case. However the goal here is to find out if this modification is already impracticable when only considering EPL students.

We run the optimal solver 20 times on the 16 data sets initially containing major/minor examinations and study the performances compared to the results obtained in Section 6.3 for these data sets.

### 6.4.2.2 Results

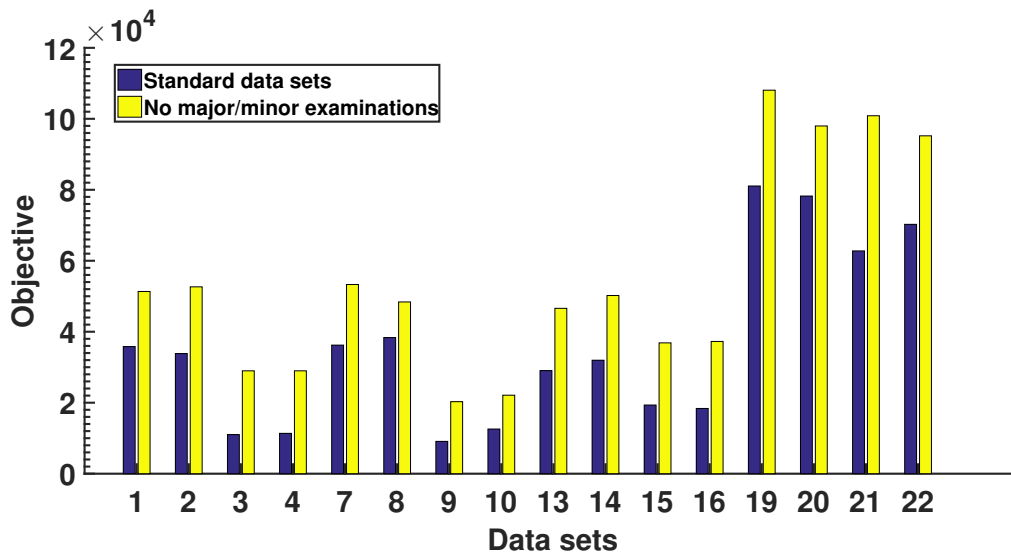


Figure 6.14: Effects of suppressing major/minor duplicates

Figure 6.14 compares the sample mean of the  $P^{BSR}$  values obtained for each data set. We observe that suppressing the major/minor duplicates is not efficient, even when only considering the EPL students. This is due to the post-processing phase allowing student to be placed at the best possibility. We can assume that if this post-processing operation is authorized, there is no gain, at least for students, to have only one choice of examination instead of two.

### 6.4.3 Evolution when adding or deleting time slots

#### 6.4.3.1 Methodology

The goal of this test is to visualize the evolution of  $P^{BSR}$  with the number of days in a session. We performed this test by running 20 times the optimal solver on 2 data sets. We observed on these 2 data sets the evolution when adding and deleting up to 4 days.

The data sets chosen were sets 7 and 17 because they do not have any fixed courses that may be scheduled at one of the deleted time slots. Set 7 contains minor/major examinations while set 17 do not.

#### 6.4.3.2 Results

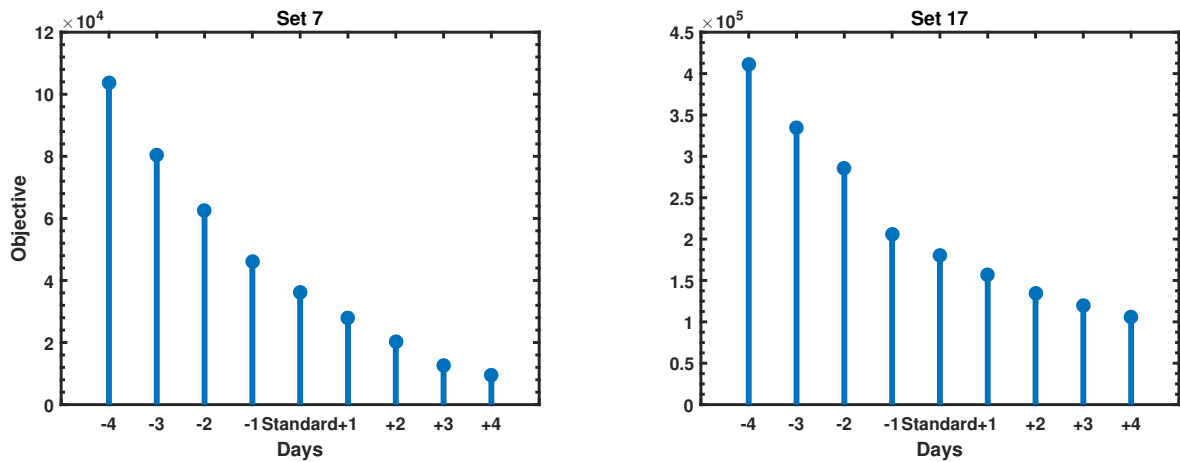


Figure 6.15: Impact of adding/deleting time slots in a session - Data sets 7 and 17

We naturally observe on Figure 6.15 that the value of  $P^{BSR}$  is decreasing less and less when the number of time slots is increasing. One could have expected a considerable drop at some point, which is not the case. This mainly shows that the gain obtained when adding a time slot is less important than the loss of deleting one time slot. So adding one day does is particularly worthy but deleting one is dreadful.

### 6.4.4 Impact of the decomposition method

#### 6.4.4.1 Methodology

In this test, we study the nested decomposition method defined in Section 4.4 on an artificial instance created for this purpose. The primal constraint graph of this instance is composed of two large connected components so that the decomposition method leads to two subproblems of almost equal size. For the sake of simplicity, we applied only this method on CILP because we can actually compute subproblems for this linear program, as we did not compute any method to extract the connected components for a

nonlinear program.

Note that the pre-processing builds a better instance for decomposition methods. By minimizing the number of conflicting pairs of courses we will sparse the conflict matrix  $C$ . There will be less interactions between the courses and we hope to create more connected components in the primal constraint graph and to make more courses penalty avoidable.

#### 6.4.4.2 Results

When running for 3 hours on the two subproblems and then merging the solutions we obtained a value  $P^{GD} = 2880$ , while solving the complete problem led to  $P^{GD} = 3564$ . We can observe that this decomposition method is useful to minimize  $P^{GD}$  if the primal graph contains multiple connected components.

## 6.5 Comparison with a real schedule

In this section, we compare how our solver performs with respect to a real schedule. We will directly use the two KPI's approaches defined in Chapter 3 and some of the properties defined in Chapter 4 to visualize by how much we can outperform a schedule done manually.

### 6.5.1 Methodology

We compared the solution produced by the optimal solver against the schedule proposed by the secretary for the session of June 2017. In this case, we use a data set that almost perfectly corresponds to reality, in comparison of the limits enunciated in Section 6.1 for the previous data sets :

1. The examinations formats and the capacity limits of orals are respected.
2. Examinations that are not proposed by EPL are scheduled at their real fixed time slot.
3. Unavailabilities are not considered, as it was not in use for this session.

The only source of divergence with reality is the fact that we do not know at which oral duplicates or major/minor examination students were enrolled. For this reason we applied the post-processing step to the real schedule to smooth the impact of this lack of information.

We believe then that the example that follows reflects properly the efficiency of the solver developed in this Thesis.

## 6.5.2 Results

We observe on Figure 6.16 that the real schedule produces much more low lexicographical gaps than the one generated by the optimal solver. The distribution of the gaps is appropriate as it matches the definition we provided in Section 3.3.1. This shows that the solver avoids to schedule examinations with many students in common in a short time frame and performs better than the manual schedule.

### Gaps distribution

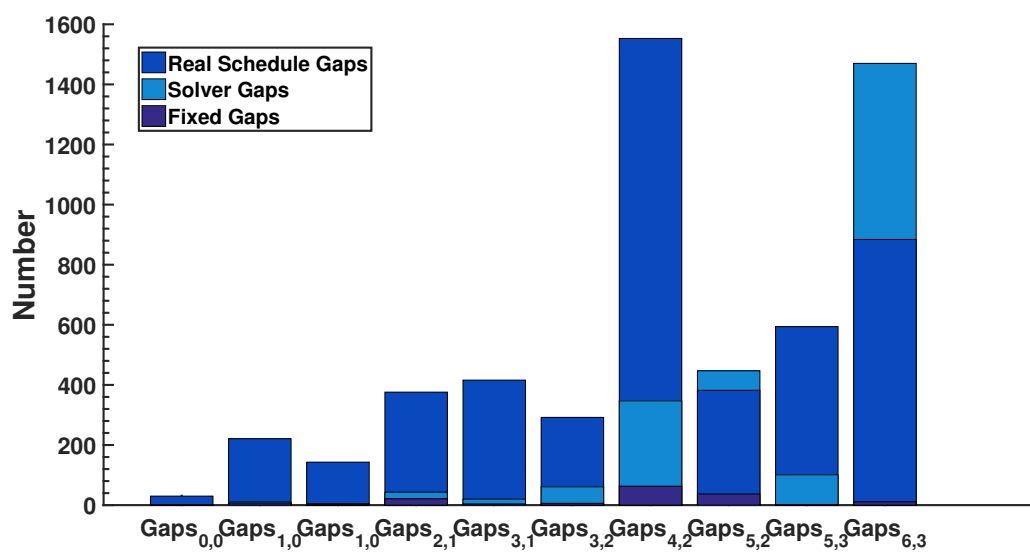


Figure 6.16: Comparison of the gaps distribution

We can also evaluate how well the solver performed by comparing the quantity of unwanted gaps with invariants defined in Section 4.5. On Figure 6.17 we observe that the sum of gaps for a subset of gaps of the schedule returned by the solver and the minimal quantity of gaps for the same subset are almost identical for small subset of gaps.

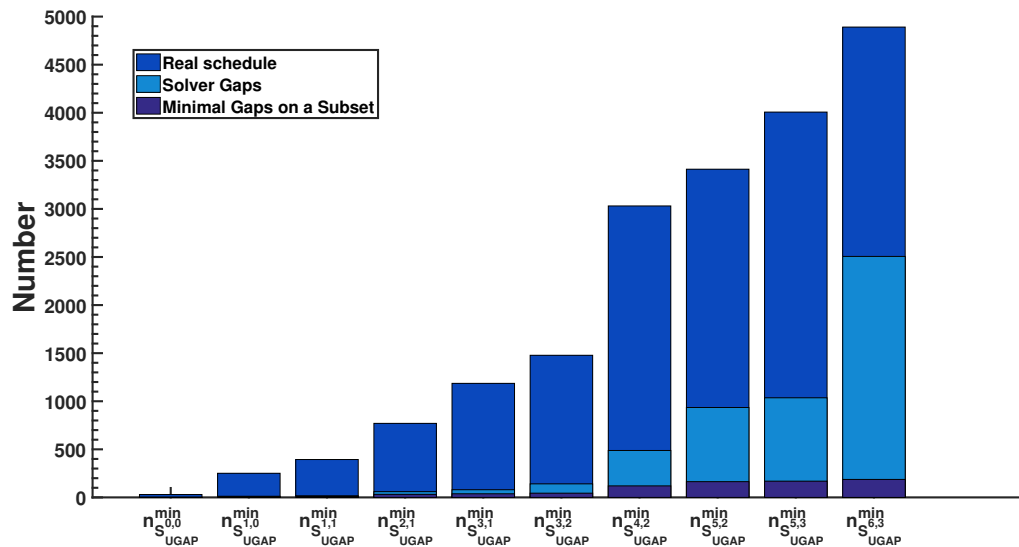


Figure 6.17: Comparison of the sum of gaps on subset of gaps with the minimum quantity of gaps on the same subset for different subset of gaps

### Bad student rate

The solver produces a schedule with BSR = 0,4 %, while the real schedule has a bad student rate of 36 %. This shows that approximatively one third of the students had to deal with one of the bad situations defined in Section 3.3.2, while it only concerns less than 1 student over a hundred if the schedule is computed using the solver.

## 6.6 Summary

This chapter shows that the solver, when parameters are rightly adjusted, is efficient for solving UCT-EPL on regular instances. We conclude in the next chapter by summarizing the achievements of the Thesis and we provide tracks of improvements.



# CONCLUSION

---

The Thesis main purpose has been set on a highly practical ambition: create an automated system generating efficient examinations schedules that can be used by the EPL administrative support. As illustrated in the tests chapter, the solver developed produces satisfying results on representative instances. It generates a timetable respecting the current administrative constraints and is satisfactory in the sense of the key performance indicators defined.

In addition to this, we ensure that the two initial subsidiary goals have been fulfilled. We explored some modifications on the administrative constraints that may offer improved schedules. Furthermore, we developed a prototype of the tool implementing the solver and granting practical graphical interface. This prototype allows to load easily the data requirements using Excel<sup>®</sup> sheets and generates schedules that can be visualized in an HTML format and saved in XML. Anyone who wishes to have access to this prototype can contact the authors <sup>1</sup>.

Besides its essential down-to-earth goals, this Thesis also presents interesting tracks to explore in further researches in university timetabling. Whereas most of the papers nowadays are getting away from integer linear programming in favor of heuristics, the FAS algorithm applied on an ILP model is one of the decisive steps of the solver. We also developed decomposition and invariants computing methods that may be easily generalized to other timetabling problems. Finally we showed once more the good results obtained when using meta-heuristic methods by the mean of the IVNS algorithm.

We isolated two main improvement tracks from the current state of work. First, the specifications and models have been defined only considering the examinations and time slots in the UCT-EPL. A model taking the rooms into account can be implemented to ease even more the task of the schedule planners. Moreover, based on the good results obtained in the complementary tests Section 6.4, the nested decomposition method

---

<sup>1</sup>charles.ninane@student.uclouvain.be & adrien.brogniet@student.uclouvain.be

might be added to the current solver to increase the performances.

The overall aspiration of this Thesis is to convince that automated scheduling tools are worthy today, especially if the schedules are currently made by hand. It saves a considerable amount of time and may generate much more satisfactory schedules for students.



# MATHEMATICAL PROOFS

---

## A.1 Penalty avoidable decomposition method

In this section we prove that the gain in worst case complexity of this decomposition method  $0 < |E_{pa}| < |E|$  and  $|T| > 2$  it follows that  $|T|^{|E|-|E_{pa}|} + |T|^{|E_{pa}|} \leq |T|^E$  and the minimum of this complexity for  $|E_{pa}|$ .

Let us first introduce the geometric formula:

$$\sum_{i=0}^{|E|-1} |T|^i = \frac{1 - |T|^{|E|}}{1 - |T|}$$

It is easily noticed that for  $0 < |E_a| < |E|$ :

$$\sum_{i=0}^{|E|-1} |T|^i \geq |T|^{|E_{pa}|} + |T|^{|E|-|E_{pa}|}$$

Now consider that  $\sum_{i=0}^{|E|-1} |T|^i = \frac{1-|T|^{|E|}}{1-|T|} \leq |T|^E$  for  $|T| > 2$  which will conclude the proof. Rearranging the terms and distributing we obtain:

$$|T|^E(|T| - 2) \geq 1$$

which is always verified for  $|T| > 2$ .

Now let us search for the minimum worst case complexity as a function of  $E_a$ . Let  $f(x)$  be a two times continuously differentiable function of  $x \in R$ . A sufficient condition for  $x^*$  to be the unique optimal solution for the unconstrained optimization problem:

$$\underset{x \in R^n}{\text{minimize}} \quad f(x)$$

is that  $f'(x^*) = 0$  and  $f''(x^*) > 0$ . Suppose now that we add constraints to the problem

and that  $x^*$  verifies the constraint. Then it is clear that in this very particular case,  $x^*$  stay optimal to the constrained optimization problem. Let us prove now that this apply to our constrained optimization problem:

$$\begin{aligned} & \underset{|E_{pa}|}{\text{minimize}} && |T|^{|E|-|E_{pa}|} + |T|^{E_{pa}} \\ & \text{subject to} && 0 < |E_{pa}| < |E| \end{aligned}$$

We have:

$$f'(|E_{pa}|) = \ln(|T|) (|T|^{|E_{pa}|} - |T|^{|E|-|E_{pa}|})$$

which leads to  $f'(|E_{pa}|) = 0$  when  $|E_{pa}| = \frac{|E|}{2}$ . and

$$f''(|E_{pa}|) = |T|^{-|E_{pa}|} \ln^2(|T|) (|T|^E + |T|^{2|E_{pa}|})$$

and thus  $f''\left(\frac{|E|}{2}\right) > 0$  and since  $\frac{|E|}{2}$  verifies the constraint, i.e  $0 < \frac{|E|}{2} < |E|$  it is the unique optimal solution of the unconstrained optimization problem.

- Contact information :

# BIBLIOGRAPHY

---

- [ABM07] Salwani Abdullah, Edmund K. Burke, and Barry McCollum. *Using a Randomised Iterative Improvement Algorithm with Composite Neighborhood Structures for the University Course Timetabling Problem*. Metaheuristics, pages 153–169, 2007.
- [BA07] M Akif Bakir and Cihan Aksop. *A 0-1 Integer Programming Approach to a University Timetabling Problem*. Hacettepe Journal of Mathematics and Statistics, Volume 37, pages 41–55, 2007.
- [Bes07] Christian Bessière. *Handbook of genetic algorithms*. Springer, Berlin, 2007.
- [BEW94] Edmund K. Burke, David G. Elliman, and Rupert Weare. *A genetic algorithm based university timetabling system*. East-West Conference on Computer Technologies in Education, pages 35–40, 1994.
- [BJKW97] Edmund K. Burke, Kirk Jackson, Jeff H. Kingston, and Rupert Weare. *Automated University Timetabling: The State of the Art*. The Computer Journal, Volume 40, pages 565–571, 1997.
- [BNW96] Edmund K. Burke, James P. Newall, and Rupert F. Weare. *A Memetic Algorithm for University Exam Timetabling*. Practice and Theory of Automated Timetabling, pages 241–250, 1996.
- [Bre76] Jon A. Breslaw. *A linear programming solution to the faculty assignment problem*. Socio-Economic Planning Sciences, Volume 10, pages 227–230, 1976.
- [CK95] Tim B. Cooper and Jeffrey H. Kingston. *The Complexity of Timetable Construction Problems*. Practice and Theory of Automated Timetabling, pages 281–295, 1995.
- [Dav93] L. Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, 1993.

- [DBH04] S. Daskalaki, T. Birbas, and E. Housos. *An integer programming formulation for a case study in university timetabling*. European Journal of Operational, Volume 153, pages 117–135, 2004.
- [DBS06] Marco Dorigo, Mauro Birattari, and Thomas Stützle. *Ant Colony Optimization*. IEEE Computational Intelligence Magazine, Volume 1, pages 28–39, 2006.
- [DM01a] M. Dimopoulou and P. Miliotis. *Implementation of a university course and examination timetabling system*. European Journal of Operational Research, Volume 130, pages 202–213, 2001.
- [DM01b] Elizabeth D. Dolan and Jorge J. Moré. *Benchmarking Optimization Software with Performance Profiles*. Mathematical Programming, Volume 91, pages 201–213, 2001.
- [Dor99] Marco Dorigo. *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. IEEE Transactions on Evolutionary Computation, Volume 1, pages 53–66, 1999.
- [DOW55] George Bernard Dantzig, Alexander Orden, and Philip Wolfe. *The generalized simplex method for minimizing a linear form under linear inequality restraints*. Pacific Journal of Mathematics, pages 183–194, 1955.
- [Egl90] Richard W. Eglese. *Simulated Annealing: A tool for Operational Research*. European Journal of Operational Research, Volume 46, pages 271–281, 1990.
- [Glo90] Fred Glover. *Tabu Search - Part I and II*. ORSA Journal on Computing, Volume 1 and 2, 1989-1990.
- [HK03] Limin Han and Graham Kendall. *An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm*. Evolutionary Computation, Volume 3, 2003.
- [Lew08] Rhydian Lewis. *A Survey of Metaheuristic-based Techniques for University Timetabling Problems*. OR Spectrum, Volume 30, pages 167–190, 2008.
- [MH97] N. Mladenovic and E Hansen. *Variable Neighborhood Search*. Computers and Operations Research, Volume 24, pages 1097–1100, 1997.
- [Mit02] John E. Mitchell. *Branch-and-Cut Algorithms for Combinatorial Optimization Problems*. Handbook of applied optimization, pages 65–77, 2002.
- [PB10] N. Pillay and W. Banzhaf. *An informed genetic algorithm for the examination timetabling problem*. Applied Soft Computing, Volume 10, pages 457–467, 2010.

- [QBML08] Rong Qu, Edmund K. Burke, Barry McCollum Liam T.G. Merlot, and S.Y. Lee. *A survey of search methodologies and automated system development for examination timetabling*. Journal of scheduling, Volume 12, pages 55–89, 2008.
- [RC95] Peter Ross and Dave Corne. *Comparing Genetic Algorithms, Simulated Annealing, and Stochastic Hillclimbing on Timetabling Problems*. Evolutionary Computing, pages 94–102, 1995.
- [RDSB<sup>+</sup>02] Olivia Rossi-Daria, Michael Sampels, Mauro Birattari, Marco Chiarandini, Marco Dorigo, Luca M. Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stützle. *A Comparison of the Performance of Different Metaheuristics on The Timetabling Problem*. Practice and Theory of Automated Timetabling IV, pages 329–351, 2002.
- [Sch99] Andrea Schaerf. *A Survey of Automated Timetabling*. Artificial Intelligence Review, Volume 13, pages 87–127, 1999.
- [SSM03] Krzysztof Socha, Michael Sampels, and Max Manfrin. *Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art*. Applications of Evolutionary Computing, Volume 13, pages 334–345, 2003.
- [Tri84] Arabinda Tripathy. *School Timetabling : A Case in Large Binary Integer Linear Programming*. Management Science, Volume 30, pages 1473–1489, 1984.
- [Woo69] DC Wood. *A technique for coloring a graph applicable to large scale timetabling problems*. The Computer Journal, Volume 12, pages 317–319, 1969.
- [WP67] Jame Anthony Dominic Welsh and MB Powell. *An upper bound for the chromatic number of a graph and its application to timetabling problems*. The Computer Journal, Volume 10, pages 85–86, 1967.

