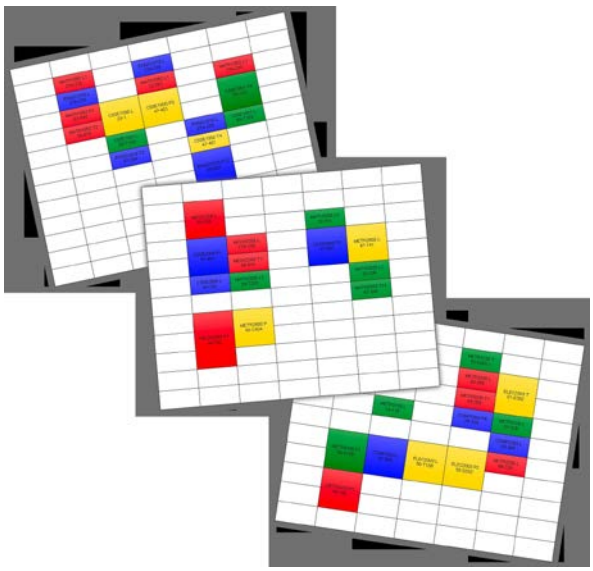
Par **Taha ARBAOUI**

Modeling and solving university timetabling

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 10 décembre 2014
Spécialité : Technologies de l'Information et des Systèmes

D2167

Thesis submitted for the degree of Doctor
Université de Technologie de Compiègne

❧

Département Génie Informatique
HeuDiaSyC, UMR CNRS 7253

Discipline: TIS

Modeling and Solving University Timetabling

By Taha Arbaoui

Jury:

Jean-Paul Boufflet	Université de Technologie de Compiègne	Supervisor
Jacques Carlier	Université de Technologie de Compiègne	Examiner
Slim Hammadi	École Centrale de Lille	Examiner
Aziz Moukrim	Université de Technologie de Compiègne	Supervisor
Eric Pinson	Université Catholique de l'Ouest	Reporter
Nikolay Tchernev	Université d'Auvergne	Reporter

Thesis Defense: 10/12/2014

Université de Technologie de Compiègne

Abstract

Title: Modeling and Solving University Timetabling

This thesis investigates university timetabling problems. These problems occur across universities and are faced each year by the practitioners. We propose new lower bounds, heuristic approaches, mixed integer and constraint programming models to solve them.

We address the exam timetabling and the student scheduling problem. We investigate new methods and formulations and compare them to the existing approaches. For exam timetabling, we propose an improvement to an existing mixed integer programming model that makes it possible to obtain optimal solutions. Next, lower bounds, a more compact reformulation for constraints and a constraint programming model are proposed. For the exam timetabling problem at Université de Technologie de Compiègne, we designed a memetic approach. Finally, we present a new formulation for the student scheduling problem and investigate its performance on a set of real-world instances.

Keywords. Timetabling, Heuristics, Integer Programming, Exact Approaches.

Supervisors: Aziz Moukrim and Jean-Paul Boufflet

Université de Technologie de Compiègne

Résumé

Titre : Modélisation et résolution de problèmes d'emploi du temps d'universités.

Cette thèse s'intéresse aux problèmes d'emploi du temps d'universités. Ces problèmes sont rencontrés chaque année par les utilisateurs. Nous proposons des bornes inférieures, des méthodes heuristiques et des modèles de programmation mixte en nombres entiers et de programmation par contraintes.

Nous traitons le problème d'emploi du temps d'examens et celui d'affectation des étudiants. Nous proposons de nouvelles méthodes et formulations et les comparons aux approches existantes. Nous proposons, pour le problème d'emploi du temps d'examens, une amélioration d'un modèle mathématique en nombres entiers qui permettra d'obtenir des solutions optimales. Ensuite, des bornes inférieures, une formulation plus compacte des contraintes et un modèle de programmation par contraintes sont proposés. Pour le problème d'emploi du temps d'examens à l'Université de Technologie de Compiègne, nous proposons une approche mémétique. Enfin, nous présentons un modèle mathématique pour le problème d'affectation des étudiants et nous étudions sa performance sur un ensemble d'instances réelles.

Mots clés. Emploi du temps, Approches heuristiques, Programmation en nombres entiers, Approches exactes.

Directeurs de thèse. Aziz Moukrim et Jean-Paul Boufflet.

To all the ones I love . . .

Acknowledgements

In the name of God, Most Gracious, Most Merciful.

I would like to thank my remarkable supervisors Prof. Aziz Moukrim and Dr. Jean-Paul Boufflet for their guidance, support and continuous enthusiasm and interest. I am also very grateful and extend my sincere thanks to members of the jury Prof. Jacques Carlier, Prof. Slim Hammadi, Prof. Eric Pinson and Prof. Nikolay Tchernev for their acceptance to be member of the jury and for their rewarding discussion and helpful comments. Without forgetting to thank the awesome teams of the Direction des Systèmes d'Informations (DSI) and the Direction à la Formation et à la Pédagogie (DFP) for their support, help and precious time to finish this work.

I would like to express countless gratitude to my family, my father, my mother, my brothers and sister and all the family member for their support and sacrifices, they have put to help me. To all my friends and colleagues, I am thankful for all the time and effort you have made for me to support me achieving my goals.

Many thanks to the rest of all the Heudiasyc members for their friendship and support to be able to finish this work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Companies, schools, universities or hospitals are regularly brought to solve a timetabling problem. Building a good timetable is often a time-consuming task that requires a considerable effort in order to meet the requirements and the needs. Flight timetables, for instance, are very crucial for the airline companies as well as airports and passengers because of the tight schedules and the various uncertain conditions airplanes operate in. Train schedules, inside or outside the cities, confront different circumstances such as peak and off-peak times, construction zones and drivers' availability. Having a good timetable is important for the students in schools and universities to be able to achieve their best results. Nurses' efficiency in hospital is influenced by the quality of the timetable they get for their shifts.

Within all these categories of problems, the university timetabling problem has gained an increasing interest in the last two decades. Due to the growing number of institutions and students across the globe, planning a university timetable has become harder than ever. Institutions and universities dispose of different environments and working places, which implies various constraints and real cases to be solved. Researchers throughout the years tried to bring forward different solution methods and procedures to face the increasing difficulty of these problems.

Among university timetabling problems, exam and course timetabling are the two most studied sub-problems. Every university faces these two sub-problems at least twice a year and is therefore brought to build a course and an exam timetable. Other variants such as student scheduling and school timetabling constitute other promising variants. While meeting the *hard constraints* is agreed on to be the condition to consider a solution feasible, it is often discussed what makes a solution "good" enough. The community often defines the *soft constraints* as the means for measuring the solution quality. However, due to the difference in both hard and soft constraints set by the institutions, it has become hard to measure the algorithms proposed in the literature. To overcome this difficulty, benchmarks

and timetabling competition were set in order to reduce the gap between research and practice and put a common base on which algorithms can be tested and evaluated. The *Toronto* benchmark, the first and the second international timetabling competitions present one of the most studied benchmarks in exam and course timetabling.

We are interested in this thesis in proposing new methodologies and approaches for course and exam timetabling in universities. Chapter 1 presents, in the first half, an introduction to the different optimization problems related to university timetabling. The second half is concerned with giving the reader a background on the existing university timetabling problems and the several methods previously proposed to solve them.

Following the presentation of the field of the study of the thesis, we detail in Chapter 2 an improved mathematical model for the exam timetabling problems. A preprocessing procedure that deduces hidden conflicts and dependencies between the exams is presented. The preprocessing is based on a transitive closure applied on the exam-related constraints such as the coincidence and precedence to deduce new ones. A small MIP model that exploits the room capacities is used to determine the exams that cannot be placed together in certain periods. To improve the existing model proposed by McCollum et al. (2012), we propose a set of reformulations and valid inequalities to help accelerate solving the different instances. We show that our models succeed in reducing the number of constraints and is able to be run for more instances than the original one. The valid inequalities proposed include cliques and the dual-feasible functions that are usually applied on the Bin Packing problems. Both models were tested on the instances of the second International Timetabling Competition (ITC2007) and the Yeditepe instances. The results show that our improved model is able to obtain better results compared to the original one.

In the continuity of Chapter 2, we present in Chapter 3 lower bounds techniques, reformulation for the linear model and a constraint programming model for the same problem. The lower bounds techniques use the structure of the instances and the cliques to assess inevitable costs. A set covering linear model is applied on a clique of exams to detect the costs implied on these exams. The new reformulations are concerned with three types soft constraints and can be generalized to any spacing constraints. They reduce the number of constraints

and variables to allow the model to run in limited memory space for the large instances. The constraint programming model presents new solutions and reduces the existing gap between the lower and upper bounds. Compared to the linear model, the CP model appears to be promising and further improvements are to be investigated.

We address in Chapter 4 the exam timetabling problem in Université de Technologie de Compiègne (UTC) and give a memetic approach to solve it. We start by the describing the problem and show that some constraints fall in the potential extensions of the second international timetabling competition given by the organizers. We then give a mathematical formulation to formally model the problem and use this formulation to validate the solutions obtained by the memetic approach. This memetic approach, which operates on a population of chromosomes, is afterward detailed. The population is represented by a set of chromosomes, each refers to a neighborhood of solutions. An indirect encoding for the chromosomes is used. To decode the chromosomes, a fast *first fit* decoding process is applied. Using the indirect encoding allows each chromosome to cover a neighborhood of solutions and therefore a better exploring of the search space is made. Hill Climbing, Light Destruction Construction and Swap are the three different operators used to help improve the chromosomes. To test the approach, we modeled and formatted a set of instances from the read-world data provided by the practitioner. The algorithm helped solve all the different instances of the university and presented a very good results compared the ones resulted from the old method. The instances were put into the ITC2007 format and are made available for the community to enlarge the existing collection of instances.

For the course timetabling, we describe in Chapter 5 the student scheduling problem in our university. To solve this problem, we propose a mathematical model with a set of preprocessing and valid inequalities. Before processing the model, a preprocessing procedure that helps reduce the number of groups and detect infeasible timetables for students is applied. Following the preprocessing, the mathematical model that aims at maximizing the number of students totally assigned is presented. We next present the valid inequalities used. The results show that the preprocessing and the valid inequalities help reduce the run time of the model and show that the optimality can be reached in a very short time.

Finally, the different parts of the thesis are discussed in a general conclusion

and future works and perspectives are presented.

CHAPTER 1

FIELD OF STUDY

Contents

1.1 Introduction

This chapter is an introduction to combinatorial problems. We first introduce combinatorial problems. Next, a non-exhaustive list of different techniques used to solve them is presented. We next provide a general introduction to timetabling

problems that are the subject of this manuscript. Finally, a conclusion of the chapter is given.

1.2 Combinatorial Optimization

This section tackles first the definition of a Combinatorial Optimization Problem (COP). Notions and terminology of graphs, algorithms and complexity are exposed. The goal is to state the main definitions and the framework of the manuscript.

1.2.1 Combinatorial Optimization Problems

An optimization problem (Pardalos and Resende [2002]) consists in finding the best variable values according to a given objective function while respecting a set of constraints. The problem's difficulty depends on the nature of the variables, *discrete or continuous*. When the variables are discrete, the problem is called a *combinatorial optimization problem* (Papadimitriou and Steiglitz [1998]). Definition 1 gives a formal description of a combinatorial optimization problem (COP).

Definition 1 *A combinatorial optimization problem* $\Phi = (\Omega, f)$ *is defined as:*

- *A set of variables* $X = \{x_1, \cdots, x_n\}$
- *Each variable* x_i *is associated to a domain* D_i*, i.e.* $x_i \in D_i$
- *A set of constraints linking the variables*
- *An objective function to minimize (or to maximize):* $f : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}$

Set Ω is called the search space. A feasible solution s for problem Φ is an element $s \in \Omega$ such that $s = \{v_1, \cdots, v_n \mid v_i \in D_i \text{ and all the constraints are satisfied}\}$. Solving a COP with an objective function to minimize consists in finding a solution s^* such that $\forall s \in \Omega, f(s^*) \leq f(s)$ (or $f(s^*) \geq f(s)$ in case of maximization).

Section 1.4 provides some examples of combinatorial optimization problems. The brute force way to solve a COP is by enumerating all the possible solutions. However, enumerating impractical and ineffective if the number of possible solutions is huge. Thus, effective and efficient techniques become a necessity for solving Φ if Ω is huge.

1.2.2 Complexity Theory

The formal definition of an algorithm is introduced by Alain Turing (Turing [1937]). This definition is deduced from the notion of a formal language for an abstract machine, called *Turing's Machine*. In a nutshell, an algorithm can be defined as: Algorithm A is a finite series of instructions allowing the user to solve a defined problem Φ. For instance, to solve a COP, an algorithm would represent all the steps needed to find a solution s^*. An algorithm is a *heuristic approach* when it does not guarantee finding solution s^* but a solution s that is as close as possible to solution s^*, i.e. $\mid f(s^*) - f(s) \mid < \epsilon$ with ϵ being a very small number.

There exist numerous methods to measure the performance of algorithm A:

- Runtime: the time spent to find solution s^*
- Memory: the memory space required to run it
- The quality of the solution
- Robustness: the capacity of the algorithm to adapt to problem's input changing

Note that runtime is dependent on the machine used. The theory of *complexity* has been introduced to measure the performance of an algorithm without considering the speed of the machine on which it is executed. The idea is to state asymtotic bounds using the size of the input data parameters.

Definition 2 *The algorithmic complexity C_A of an algorithm A is defined as the number of instructions needed to solve any instance of problem Φ of size n.*

The Landau's notation defines for instance asymptotic lower or upper bound for C_A. The notation $\mathcal{O}$ corresponds to an upper bound of C_A: A is said to be of complexity $\mathcal{O}(g(n))$ if $\exists M > 0, \exists n_0$ such that $\forall n > n_0, C_A \leq Mg(n)$. An algorithm A is said to be polynomial if $g(n)$ represents a polynomial function. Below, some examples of algorithmic complexity:

- $\mathcal{O}(1)$: constant complexity independent of the size of Φ
- $\mathcal{O}(logn)$: logarithmic in the size of Φ
- $\mathcal{O}(n)$: linear in the size of Φ

- $\mathcal{O}(n^p)$: (with $p \geq 2$) polynomial in the size of Φ
- $\mathcal{O}(a^n)$: (with $a \geq 2$) exponential in the size of Φ

The complexity can also depend on some parameters of the problem. For example, $g(n) = n^p v^q$ with v being a parameter of the problem and q a constant. The algorithm in this case is called *pseudo-polynomial*.

1.2.3 Complexity of Decision Problems

Considering that solving complex problems in general and optimization problems in particular is a source of attraction for several researchers, computer performance has gained a noticeable advancement. For some optimization problems, no one has yet found a polynomial algorithm to solve them. For others, we know that we can solve them with a polynomial algorithm. Thus, a natural classification would be classifying the problems according to the complexity of algorithms solving them (Garey and Johnson [1979]).

To introduce the notion of complexity classes of problems, we introduce the definition of decision problems: a decision problem is a problem to which the answer is either a *yes* or a *no*. Each optimization problem with an objective function f disposes of an equivalent decision problem. The decision problem would be formulated as follows: considering $k \in \mathbb{R}$, does there exist a solution s^* for which $f(s^*) = k$?

Definition 3 *A problem is in class P (polynomial time) if a polynomial algorithm that solves it exists.*

Definition 4 *A decision problem is in class NP (non-deterministic polynomial time) if verifying that the solution is valid can be done in a polynomial time.*

Definition 5 *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.*

Definition 6 *An optimization problem is said to be NP-Hard if its decision problem is NP-Complete.*

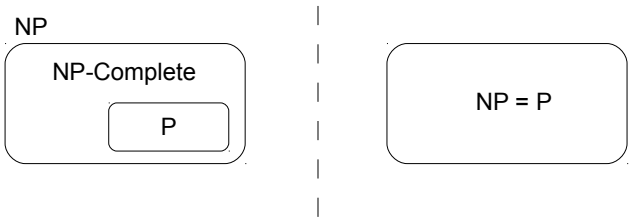


Figure 1.1 – P vs NP

Definition 7 *An optimization problem (or decision) is NP-Hard (or NP-Complete) in the ordinary sense if it is NP-Hard (or NP-Complete) and if there exists a pseudo-polynomial algorithm to solve it.*

As a result, we easily notice that $P \subseteq NP$. The remaining question to answer (and to prove) is: if $P \subset NP$ or $P = NP$. If $P = NP$ then the set of NP-Complete problems is empty, which implies that there are always polynomial algorithms that solve NP problems. This hypothesis, difficult to accept, lead the researchers to orient their works on proving that $P \neq NP$. For a new optimization problem, in order to prove that it is NP-Complete, a polynomial reduction can be used (Karp [1972]).

1.2.4 Graph classes

A graph is a mathematical object that uses nodes to model elements of the problem and edges (or arcs) as relationships between the elements. Graphs represent a powerful tool, used to model and solve combinatorial optimization problems. These definitions will be used to describe approaches developed for the timetabling problems.

Definition 8 *A graph G is defined by a set of nodes V and a set of edges $E \subseteq V \times V$ that represents the relationships between the nodes. The graph is complete if $E = V \times V$. The neighborhood of a node i is defined as: $N(i) = \{j \mid (i,j) \in E\}$ and the degree of node i is $\mid N(i) \mid$.*

When there is no orientation mentioned for the edges, the graph is called *a non-oriented graph*. If orientations exist, we have a *directed graph*.

Definition 9 *A path in a directed graph G is a list of nodes $(v_1, \cdots, v_k)$ such that $(v_i, v_{i+1}) \in E, \forall i < k$. If $v_1 = v_k$, the path is called a directed cycle.*

Definition 10 *$G' = (V', E')$ is a sub-graph of graph $G = (V, E)$ if it is obtained by removing nodes or edges, i.e. $V' \subseteq V$ and $E' \subseteq E$.*

Definition 11 *$G(S)$ is an induced graph of $G(V, E)$ if $S \subseteq V$ and $\forall\, i, j \in S, \quad (i, j) \in E$ is kept from the original graph.*

Definition 12 *A **Clique** is a subset of nodes $S \subseteq V$ such that $G(S)$ is a complete graph*

Definition 13 *An **independent set** of G is a subset of nodes $I \subseteq V$ such that $G(I)$ does not contain any edge*

Definition 14 *A **vertex cover** is a subset of nodes $S \subseteq V$ such that the set $E_S = \{\forall (i, j) \in E, i \in S \text{ or } j \in S\}$ is equal to E*

Definition 15 *A **coloration** of graph G consists in assigning to every node i of the graph a color c_i such that $c_i \neq c_j, \forall (i, j) \in E$. The minimum number of colors needed to color graph G is called the **chromatic number**, denoted $\chi(G)$.*

1.3 Solution Approaches

We provide next a general idea on some of the different methods used to solve COPs. Prior to applying any approach to solve a COP, preprocessing can sometimes be applied so as to reduce the size of a problem or to discover dependencies useful to facilitate the processing. We first give an example of preprocessing. Heuristic and metaheuristic algorithms are described next. Finally, exact algorithms such as tree search and mixed integer programming are introduced.

1.3.1 Preprocessing

Preprocessing is a technique used to modify the problem in order to ease the solving process. Preprocessing usually reduces the search space of the problem by analyzing the data of the problem and its nature, namely the objective function and the constraints. For instance, by analyzing the constraints of a COP Φ, some

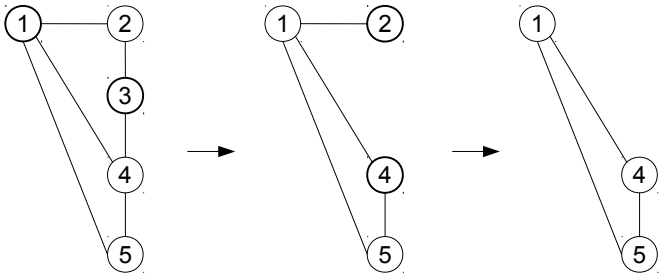


Figure 1.2 – Preprocessing example

variables x_i can be fixed or the size of D_i can be reduced. However, we have to guarantee that the optimal solution for the reduced problem must be the optimal for the original one.

Let us consider a concrete example, finding the maximum clique in a graph G. The following preprocessing will allow us to reduce the size of graph G: if $\exists(i,j) \in E$ such that $N(j) \subset N(i)$, then the node j can be deleted from graph G. The reason is that a maximal clique K_j (i.e. $\forall k \in V \backslash K_j, K_j \cup \{k\}$ is not a clique) containing node j has a size that is always less or equal to the size of the maximum clique K_i containing node i. Figure 1.2 illustrates a series of this preprocessing on an initial graph which gives eventually the maximum clique: we have $N(3) \subset N(1) \Rightarrow$ node 3 can be deleted. Next, node 2 is deleted because $N(2) \subset N(4)$. The remaining nodes in the graph represent the maximum clique of the initial graph.

1.3.2 Heuristic Algorithms

The heuristic algorithms to solve NP-Hard problems can be divided into two categories: PTAS and non-PTAS algorithms. The first category involves the algorithms with polynomial complexity that assures a certain distance from the optimal solution in the worse case. These algorithms are called *PTAS* (*Polynomial-Time Approximation Scheme*). The second category of algorithms produces generally a very good solution in a short time, even though no distance from the optimal solution is guaranteed.

Greedy Algorithms

Greedy algorithms (**Cormen01**) are simple *constructive algorithms*. These algorithms takes a partial (or an empty) solution and fill it element by element. Elements are incorporated in turn in the solution. They are chosen according to a greedy criterion. These algorithms does not question the decision taken in previous iterations. The resulting solution is most likely not optimal.

Despite their simple design, the greedy algorithms are useful when integrated in a heuristic approach. They are widely used as a part of elaborated approaches, especially in preparing solutions for local searches. They are easily maintained and flexible when changing the criteria.

Local Searches

Local searches (Hoos and Stutzle [2004]) are often effective algorithms used to solve COPs. Most approaches developed to practically solve NP-Hard COPs contain one or more local searches. Technically, a local search uses a current solution s and repeatedly replaces it with a better solution s' close to s in the search space. The set of solutions close to solution s is called *Neighborhood of s*, noted $N(s)$.

Definition 16 *A neighborhood structure is a function N:* $\Omega \rightarrow 2^{\Omega}$ *which is associated to a solution* $s \in \Omega$ *where* Ω *is the search space and* $N(s) \subseteq \Omega$.

In general, a neighborhood N is not defined by a partial enumeration of Ω but by an operator of transformation of s. Ideally, local search algorithms permit to start with a solution $s \in \Omega$ and, using a series of transformation, to reach the optimal solution s^* of the problem. When developing a local search, it is usually recommended that the comparison between $f(s)$ and $f(s')$ can be done in a short time. A smart and fast local search avoids comparing $f(s)$ and $f(s')$.

Definition 17 *A local optimum of a neighborhood* $N(s)$*, noted* s^*_N*, of a minimization problem (resp. maximization) is a solution that verifies* $s^*_N \in N(s)$ *and* $\forall s' \in N(s)$, $f(s') \geq f(s^*_N)(resp. f(s') \leq f(s^*_N))$. *This optimum is strict if* $f(s') > f(s^*_N)$ $(resp. f(s') < f(s^*_N))$.

There are two ways of choosing s' out of $N(s)$: the first one is by selecting s^*_N (best improvement) whereas the second consists in taking the first solution $s' \in N(s)$ that has strictly a better quality than s (first improvement).

Other local searches proceed otherwise. They permit developing complex metaheuristics or obtain solutions with the same quality. Two examples of these local searches are:

- *Random Walk:* s est replaced by s' which is not always better. This allows the local search to escape from local optimum. If the solutions accepted are only the ones with the same quality as s, the local search is then called a *plateau search*
- *Stochastic local search (Hoos and Stutzle [2004]):* some of the solutions in $N(s)$ are randomly generated and the best solution s' is used to replace the solution s. If the quality of s' is lower than the one of s, the solution is accepted with a certain threshold (or probability). This technique is useful when the neighborhood $N(s)$ is large. In this category of local searches, we can mention the simulated annealing (Kirkpatrick, Gelatt, and Vecchi [1983]; Černý [1985]) and the ruin and recreate principle (Schrimpf et al. [2000])

Metaheuristics

Metaheuristics (Blum and Roli [2003]; Glover and Kochenberger [2003]) are more elaborated algorithms and represent a large part of the techniques used to solve most of the COPs. Numerous metaheuristics exist today and their conception comes from various sources of inspirations. Some are made by analogy to other scientific fields such as physics (simulated annealing), biology (ant colony and evolutionary algorithms), neurology (tabu search) and sociology (memetic algorithms, particle swarm optimization and multi-agent systems).

Our purpose is not to exhaustively present the metaheuristics used in the literature. We give outlines of algorithms relevant to this thesis hereafter. We classify these algorithms into two parts: solution-based algorithms and population-based algorithms.

Greedy randomized adaptive search procedure - GRASP (Feo and Resende [1995]): this procedure is an elaborated version of constructive heuristics with local searches. In each iteration, a new solution is generated using random elements. The solution is then improved using a local search. The best solution is saved during the process.

Iterated local search - ILS (Glover and Kochenberger [2003]): ILS consists in mixing local searches with perturbation/mutation of elements. In each iteration, the current solution s is modified and perturbed using the two operators. First, the solution is perturbed with the mutation operator. Second, it is later improved using a local search operator. If $f(s')$ is better than $f(s)$, s' replaces s and becomes the current solution. There exist several variants of ILS in the literature. Some accept worse solutions with certain conditions. Others consider more than one type of perturbations and apply them at specific steps of the algorithm.

Variable Neighborhood Search - VNS (Hansen and Mladenovi [2001]): multiple neighborhoods are used inside the local search to escape from local optimum. A solution can be a local optimum in a neighborhood but not in another. The algorithm proceeds as follows. Given the current solution s (initialized randomly using a heuristic), a set of neighborhoods $(N_1, N_2, \cdots, N_k)$ is associated with it and i the index of the current neighborhood (initialized by 1). In each iteration of the algorithm, neighborhood N_i is explored to obtain a new solution s'. If s' is better than s, then s' becomes the current solution, otherwise i is increased for the next iteration. When the solution cannot be enhanced, VNS changes the neighborhood in hope of finding a better neighborhood or diversifying the solution. This flexibility resulted numerous VNS schemes in the literature, namely *reduced VNS, skewed VNS* and *VND*.

Simulated Annealing - SA (Kirkpatrick, Gelatt, and Vecchi [1983]; Černý [1985]): by an analogy to metallurgy, *simulated annealing* is concerned with a solution s and considers the objective function of the problem as the system's energy at a given temperature T (initialized at high temperature). In each step, transformations on the current solution s are done to obtain a new solution s'. If $f(s') < f(s)$ (in case of a minimization problem), s is replaced by s'. Otherwise, the replacement is considered with a probability $e^{-\frac{f(s')-f(s)}{T}}$. Worst solutions can be accepted to allow the algorithm to escape from local optimum. The temperature is managed by a cooling strategy.

Tabu Search (Glover and Laguna [1997]): coming from a biological prospective, tabu search emulates the memory function. At a iteration t, the tabu search considers a solution s and a list of solutions recently visited L_t, called the *tabu list*. The aim of the tabu list is to memorize recently visited solutions in order to avoid re-exploring them. Additionally strategies sometimes permit to re-consider

these solutions to escape from local optimum. The algorithm explores the neighborhood $N(s)$ and chooses the solution s^*_N from the best solutions of $N(s)\backslash L_t$ to replace solution s. The tabu list L_{t+1} of the next iteration consists in list L_t with the new solution s^*_N. If the max size of L_t is reached, adding a new solution will automatically remove the oldest solution of the list. The algorithm's efficiency comes from the tabu list and in practice we do not keep the solutions in the list but their signature.

For population-based algorithms, here are some of the most known:

Genetic Algorithm - GA (Holland [1975]): Inspired by molecular biology, GA considers a solution as a *chromosome structure* containing good and bad phenotypes. Assuming that good phenotypes are part of the optimal solutions, GA uses the reproduction mechanism, the natural selection and the mutation principle to produce new solutions in which the chromosome contains better phenotypes. In practice, at each iteration, a set of chromosomes are selected for reproduction. The selected chromosomes are then crossed to produce new chromosomes called *children chromosomes*. The children are then mutated using a mutation operator with a low probability. Their *fitness* (solution quality) is then calculated and they are inserted in the population. The population considered after crossing depends on the algorithm's variant. The first variant is the *generational genetic algorithm (GGA)* in which only the new chromosomes constitutes the population. The second variant is the *steady-state genetic algorithm (SSGA)* in which the children are not directly inserted but are in competition with the existing chromosomes.

Scatter Search - SS (Glover and Kochenberger [2003]): The algorithm works on a *reference set*. The reference set represents the best solutions found. At each iteration of the algorithm, new solutions are generated randomly or with a heuristic and are later combined with the references to have intermediate solutions. The solution generated (new and intermediate) are then improved using a local search. The best of the improved solutions are inserted in the reference set. This update of the reference set has to assure diversity within the references of the population. The generalization of scatter search is called *path-relinking (PR)*. It is about a progressive combination to explore the path relating the solutions generated and the references in the reference set.

Swarm Intelligence - SI (Bonabeau, Dorigo, and Theraulaz [1999]): the swarm intelligence is a generic term for metaheuristics inspired by intelligent collec-

tive aspects. They include ant colony optimization (ACO) (Colorni, Dorigo, and Maniezzo [1992]; Glover and Kochenberger [2003]) , particle swarm optimization (PSO) (Kennedy and Eberhart [1995]), multi-agent systems (MAS) (Meignan, Koukam, and Créput [2010]; Roli and Milano [2002]), etc. We describe the PSO as an example of these metaheuristics. PSO has been developed first for the continuous-variable optimization. The main idea is to observe the orientations of animals when searching for food in groups. A particle represents an individual of a group in a swarm and the solutions are the positions (or the search places). Every particle memorizes its current position in the search space and the best position that it visited. The best individual position represents the individual experience of the particle. The global best position found by the population represents the group's experience. Every particle has a movement speed that represents the degree of change that can occur on its solution in the next iteration. At each iteration of the PSO algorithm, the speed and the current position of an individual are updated. Speed update is managed using three orientations: the current speed of the particle multiplied by w (inertia factor), the tendency of returning to the previous individual experiences multiplied by c_1 (cognitive factor) and the tendency of group's experiences multiplied by c_2 (social factor). That is, the movement's speed guides the particles to the global optimum. The application of PSO on optimization problems does not have the same success that the other population-based algorithms have had (for instance Genetic Algorithms). Indeed, a great difficulty is faced in the different parts of the algorithm that have to be adjusted according the problem's definition.

Memetic Algorithm - MA (Corne et al. [1999]): the term was introduced by Moscato (1989) and is widely used in recent works to point out the hybrid method between global search (classic metaheuristic) and local search. It has been observed that the genetic algorithm, for example, is not efficient enough for some problems (Hoos and Stutzle [2004]; Park and Carter [1995]). The reason is that mutation, crossover and selection cannot intensify the search sufficiently. Crossover and mutation play a role in diversifying the population but seem to be not efficient enough to improve the population. To overcome this, the genetic algorithm is usually hybridized with local searches. Such a scheme allows the algorithm to intensify the search in the areas explored by the genetic operators. In some memetic algorithms, the mutation stage is replaced by local searches. Moscato

(1989) compared the different memetic algorithm versions in order to analyze the performance of the different local searches.

1.3.3 Exact Algorithms

Efficiency of new heuristic approaches are often assessed by comparing the solutions they obtain to those resulted from previous approaches. A safer method is to compare the quality of the solutions obtained by the new approaches to the *theoretical bounds* or to the exact solutions.

In this section, we present the definition of theoretical bounds and two general exact approaches used to solve COPs.

Theoretical Bounds

Bounds of a COP are defined as the boundaries between which the evaluation of the optimal solution is included. Thus, the *upper and lower bounds* are used for COPs.

Definition 18 *For a COP $\Phi = (\Omega, f)$, a lower bound LB is a value such that $\forall s \in \Omega, f(s) \geq LB$. In the same spirit, an upper bound UB is defined as $\forall s \in \Omega, f(s) \leq UB$.*

For a minimization problem, solutions obtained using heuristic approaches represent upper bounds (UB), whereas a lower bound (LB) is obtained by solving a relaxed version of the problem. When $LB = UB$ for an instance of the problem, the instance is solved.

Branch and Bound Scheme

The *branch and bound scheme (B&B)* (Clausen [1997]) is a method that enumerates solutions in the search space of a COP $\Phi = (\Omega, f)$. This enumeration is done by successive branching on Ω into subspaces $(S_1, \cdots, S_k)$ and by bounding these sub-spaces. These evaluations are performed by calculating the lower and upper bounds of Ω on a sub-problem. For example, for a minimization problem, to accelerate solving the problem, we can prune a sub-space S_i if $LB(S_i) \geq UB_{best}$ where UB_{best} is the best upper bound known during the search. This pruning implies that solutions of the sub-space S_i will not be explored. When efficient

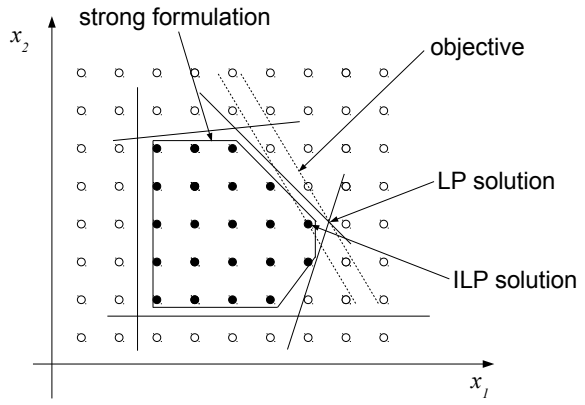


Figure 1.3 – Example of Mixed Integer Programming (MIP)

pruning strategies and evaluation can be designed, B&B is usually faster than a complete enumeration of the solutions.

The runtime efficiency of a B&B algorithm depends on the following factors:

- The bounding scheme: different techniques for bounding result in different tree searches.
- UB and LB methods: the more effective the methods that assess the UB and the LB are, the more efficient decisions, taken during the search, are.
- Exploration strategy: while branching, the strategy used to generate subspace to be explored highly contribute to the efficiency of the algorithm

Linear Programming

A linear program is a continuous COP in which the constraints and the objective function are linear. Solving a linear program is proved to be done in a polynomial time using the *ellipsoid method* Khachiyan (1979). Although not proved to be theoretically performed in polynomial time, the simplex method is widely used thanks to its efficiency in practice. Most of the commercial software (CPLEX, GUROBI, XPress, SCIP, etc.) implements this method and the interior point method (barrier method) (Dantzig and Thapa [2003]). The interior point method is also proved to be polynomial.

COPs can generally be modeled using Mixed Integer linear Programming (MIP). The integrity constraint on variables makes solving a mixed integer linear

program difficult, generally NP-Hard. Figure 1.3 shows the difference between solutions of a MIP and an LP. Due to relaxed constraints, the LP solution shown is not included in the convex hull of the strong formulation. The dots in bold represent the ILP solution and are obtained when solving the MIP problem.

Efficient techniques used to solve MIPs are the adaptation of the Branch-and-Bound scheme by integrating mathematical tools such as *cutting planes*. When the cutting planes are added to the branch-and-bound algorithm, the algorithm is then called *branch-and-cut*. This algorithm is also available on most of the software.

There are several ways to model a COP using a MIP or an LP. The model is said to be strong if the polyhedron relax the integrity constraint. The model can be strengthened using the cutting planes. Unfortunately, the number of variables can be exponential on some problems and therefore the tree search would be huge. In such a case, column generation (Desaulniers, Desrosiers, and Solomon [2005]) is used to efficiently solve the program. Integrating column generation in a branch and bound algorithm results the *branch-and-price algorithm* (Desaulniers, Desrosiers, and Solomon [2005]). Note that the Branch-and-Price scheme is a problem-dependent scheme.

1.4 Timetabling Problems

The Oxford Advanced Learner's Dictionary defines a timetable as "a list showing the times at which particular events will happen". Wren (1996) described timetabling as a special type of scheduling. He defined timetabling as follows:

> *Timetabling is the allocation, subject to constraints, of given resources to objects being placed in spacetime, in such a way as to satisfy as nearly as possible a set of desirable objectives.*

A timetabling problem is usually composed of two types of constraints: *hard and soft constraints*. Hard constraints are to be satisfied to consider the solution valid for the problem. Soft constraints can be violated and are used to assess the quality of the solution. Each violation implies a penalty on the solution that is added to the cost.

Since the early 1960's, numerous research papers reporting work on timetabling problems have appeared in the literature. Today, research in this area

is still active and new research directions are continuing to emerge. Overviews and surveys can be found in papers by Burke et al. (1997a); Werra (1985); Leung (2004).

The educational timetabling problems can be classified into three types (Schaerf [1999]), each with their own specific characteristics and constraints:

- **School timetabling:** These are problems that are concerned with assigning the weekly lessons in schools. The aim is to assign a set of teachers to a set of classes (groups of students) for a set of lessons in a set of periods, while satisfying a set of constraints. There are many variations to the basic problem. For example, in junior (lower) schools, sometimes a single teacher remains in the same room with the same class all the day, teaching a variety of subjects. In secondary schools, teachers may remain in the same room or a teacher may move between rooms for different lessons. Examples of hard constraints are: no teacher may teach at two different rooms in the same period and that no classes can have different lessons at the same time. Soft constraints may cover issues such as rest periods for teachers, teachers preferences for certain rooms and / or specific timing of certain lessons. Further examples of constraints are listed by Costa (1994).

- **Exam timetabling:** The main objective is to assign a set of exams to a given set of time slots. Each exam has a list of enrolled students. A main hard constraint is that no student can sit more than one exam at the same time. Further details on the problem specification and examples of hard and soft constraints are given in the Section 1.4.1.

- **Course timetabling:** The purpose is to assign courses and associated events, groups of students and lecturers to time slots in such a way that no conflict occurs at any period. The number of students assigned to a room should be no more than the maximum room capacity. More details are given in Section 1.4.2.

Basically, the core characteristic is to assign events to time slots while minimizing soft constraint violations. However, there are substantive differences between these problems. For example, in exam timetabling problems, two exams can take place in the same room, this is not the case in course timetabling. See Carter and

Laporte (1998) for a more detailed description of the differences between school and university course timetabling. Description of the differences between exam and course timetabling are detailed in McCollum and Ireland (2006).

1.4.1 Exam Timetabling

Carter, Laporte, and Lee (1996) defined the exam timetabling problem as:

> *The assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes.*

Conflicts or clashes are institution-dependent hard constraints. A timetable which satisfies all hard constraints is called a feasible timetable. In addition to hard constraints, there are often soft constraints whose satisfaction is desirable but not essential. The set of constraints which need to be satisfied is usually very different from one institution to another (Burke et al. [1995]). Examples of widely encountered hard constraints are the following:

- students cannot sit for two exams at the same time
- each exam should be assigned to one period and at least to a room
- room capacities should always be respected at any period

Each institution has different requirements for evaluating the quality of a feasible timetable. In many cases, the quality is assessed using a penalty function which measures the soft constraints violations. Examples of soft constraints are the following:

- Exam A shall be scheduled before/after exam B
- Avoid students having to sit exams in consecutive time slots
- Exams with a large number of students should be scheduled earlier in the timetable
- Only certain time slots and/or rooms may be available for particular exams
- Some exams should be scheduled in the same time slot

The timetabling problem, in its simplest form, can usually be modeled as a graph coloring problem. Nodes represent the exams, colors represent the time slots and the edges represent the conflict between exams (Werra [1985]; Carter, Laporte, and Lee [1996]; Burke et al. [2004b]). In the graph coloring problem, the goal is to find the minimum number of colors used to color the vertices such that no two adjacent vertices are colored with the same color. This minimum number of colors $\chi(G)$ of a graph G. If the exam timetabling problem is considered as a graph coloring problem, the aim is to find the minimum number of time slots which are able to accommodate all the exams without any clashes.

The student enrollment lists permit to build a conflict matrix $C = [c_{ij}]$ where $i, j \in \{1, ..., N\}$ and N is the number of exams. Element c_{ij} denotes the number of students enrolled for both exam i and exam j. When a non-weighted graph is employed, it is also possible to use $c_{ij} = 1$ if there is conflict between exam i and exam j; $c_{ij} = 0$ otherwise. C is a symmetrical matrix, i.e. element $c_{ij} = c_{ji}$. For diagonal cells (i.e. i = j), each cell either contains the number of students enrolled for the particular exam (c_{ij} = number of students for exam i) or the cell contains zero ($c_{ij} = 0$) to denote that there is no conflict. Either is acceptable, depending on how the information stored in the conflict matrix is used. The conflict matrix is used to build the conflict graph. The number of exams in conflict for an exam is equivalent to the node degree. Node degree values are used, for example, when heuristic orderings (e.g. Largest Degree, Largest colored Degree and Weighted Largest Degree) are used to build solutions.

The benchmarks proposed in the literature vary according to the hard and soft constraints. Carter, Laporte, and Lee (1996) proposed the *Toronto Benchmark*. This benchmark is a set of thirteen instances taken from universities around the world. The authors studied the set of instances with regards to two different objectives: a graph-coloring objective and a exam-spacing objective. For the graph-coloring objective, the aim is to plan a clash-free timetable in the minimum number of periods. Exams represent the nodes in the graph and the number of periods is the number of colors to be minimized. For the exam-spacing objective, the purpose is to build a clash-free timetable in a fixed number of periods while spacing out the exams for the same student. Burke, Newall, and Weare (1996) proposed the *Nottingham Benchmark*. The instances of this benchmark were taken from the Nottingham university. The objective to minimize was avoiding exams in the same

day for the students. Merlot et al. (2003) proposed the *Melbourne Benchmark* which is a set of two instances. The objective was to minimize the occasions of students having two exams consecutively either on the same day or overnight. In 2007, the second International Timetabling Competition (ITC2007) proposed a new benchmark. The goal of the competition was to reduce the gap between research and practice by proposing real-world instances. The objective function to minimize is a set of summed penalties related to the violation of soft constraints. The competition brought a new set of hard and soft constraints that didn't exist in previous benchmarks. An example of hard constraints is the *room exclusive* constraint. It states that some exams require to be scheduled alone in the room. An example of soft constraints is the *front load* constraint. The constraint requires some exams that are identified to be "large" to be scheduled early in the timetable. If this constraint is not respected, a corresponding penalty is applied.

Researchers continue to study the Toronto benchmark and the interest in the benchmark of ITC2007 is increasing. Even though the set of benchmarks does not represent all the constraints available in universities, it still gives a good test base on which approaches can be tested.

1.4.2 Course Timetabling

A general overview of course timetabling can be found in the paper by Carter and Laporte (1998). A complete formal description of the problem can be found in Burke et al. (2004b).

The course timetabling is defined in Carter and Laporte (1998) as:

> a multi-dimensional assignment problem in which students, teachers (or faculty members) are assigned to courses, course sections or classes; "events" (individual meetings between students and teachers) are assigned to classrooms and times.

As stated earlier, in university course timetabling, a set of courses and associated events is assigned to a set of rooms and periods within a week and, at the same time, students and teachers are assigned to the courses so that the appropriate lessons can take place, subject to a variety of hard and soft constraints. In 2002, Paechter (*Metaheuristics Network.* [2002]) introduced a course timetabling problem instance generator as part of an "International Timetabling Competition". The

objective of the first International Timetabling Competition was to create feasible weekly class timetables for a university, in which a number of hard constraints were satisfied, while minimizing the number of soft constraints violated. The instance generator was used to produce simplified, but realistic, problem instances, all of which had at least one perfect solution (a solution with no constraint violations, hard or soft).

The competition used the following hard constraints:

- No student is required to attend more than one course at the same time
- A course can only be scheduled to a room which satisfies the features required by the course
- A course can only be scheduled to a room which has enough room to accommodate all students registered for it
- Only one course can be scheduled in one room at any time slot

By definition, it is not compulsory to satisfy the soft constraints for any given problem. Thus, some form of penalty function is used to measure the degree to which the soft constraints are satisfied. There is no universally accepted method, the number of students for which each constraint is not satisfied are usually simply summed.

Automated approaches for course timetabling have been studied over the last thirty years. A comprehensive survey of the early approaches can be found in Carter and Laporte (1998). Other surveys of university timetabling that cover both examination and course problems include Burke et al. (2004a); Schaerf (1999); Wren (1996). The set of twenty instances introduced for the competition (three more instances were also generated, to be used as 'unseen' tests) have also been used by a number of authors as a benchmark data set. The competition was won by Kostuch (2005) utilising a 'three-phase approach' featuring simulated annealing, which obtained the best results on 13 out of the 20 instances. Burke et al. (2004a) used an approach based on the Great Deluge Algorithm, which obtained the best results on the remaining 7 of the 20 problem instances. Other approaches used by the competitors include those based on simulated annealing, a hybrid local search method and several variations of tabu-search.

Paechter's test instance generator was used by Socha, Knowles, and Sampels (2002) to generate eleven problem instances of various sizes. They compared a local search method and an Ant Colony Optimization algorithm on these eleven problem instances and showed that the Ant Colony Optimization algorithm achieved better performance. The same eleven problem instances have subsequently been used by other authors as a comparison basis. Burke, Kendall, and Soubeiga (2003) introduced a hyper-heuristic that utilized tabu search in an attempt to raise the level of generality of automated timetabling systems, and the system was used to solve both these course timetabling problem instances and nurse scheduling problems. Burke et al. (2007) developed a graph-based hyper-heuristic approach which used a sequence of heuristic orderings to construct the initial solution and then applied steepest descent local search to improve the solution.

These data sets were also used by Abdullah, Burke, and Mccollum (2005) who employed a variable neighbourhood search with a fixed length tabu list used to penalise the unperformed neighbourhood structures. Following on from this, Abdullah, Burke, and McCollum (2007) applied a randomised iterative improvement method featuring composite neighbourhood structures to the test instances. Despite the fact that the problem of timetabling university courses is very different from timetabling university examinations, some authors have blurred the distinctions and/or have applied the same techniques to solve both problems (McCollum and Ireland [2006]).

1.4.3 Student Scheduling

Despite the fact that course and exam timetabling represent the most encountered problems, the Student Scheduling Problem (SSP) is included, whether implicitly as in the course timetabling or explicitly by considering it a sole problem.

Cheng, Kruk, and Lipman (2003) defines the student scheduling problem as:

> The Student Scheduling Problem is the assignation of students to sections of courses offered at various times during the week. The objective is to fulfill student requests, providing each student with a conflict-free schedule (no two assigned sections meeting at the same time), while respecting room capacities and possibly also balancing section sizes (or some other side constraint).

The student scheduling problem can be considered as a part of course timetabling. Since it can be considered a part of course timetabling, it did not get the same attention the exam and course timetabling have got. The main differences between the course timetabling and student scheduling problem are that the course timetabling aims at fixing the timetable whereas the student scheduling problem disposes of a fixed timetable. The fixed timetable in the SSP contains the different courses taught in the university and is made available for the students in order to allow them choosing the different courses. The courses are composed of one or multiple sections, each of which has a timeslot and a room in the timetable.

SSP was discussed by different researchers in the literature. Laporte and Desroches (1986) provide a mathematical model for SSP. Their model consider one hard constraint which is respecting the list of preferences for the students. However, the conflicts are considered as soft constraints and thus to be minimized. The problem is solved in three phases: it starts by finding a feasible solution, balancing the sections and finally adjusting the solution to respect the room sizes. Cheng, Kruk, and Lipman (2003) refers to SSP as part of solving the American high school timetabling problem. The goal is to respect the list of student preferences while having a conflict-free timetable. They demonstrate that SSP is an NP-hard problem and present a multi-commodity flow formulation to solve. Broek, Hurkens, and Woeginger (2009) present the timetabling problem in the TU Eindhoven. They provide two problem formulations and give some complexity results to the different variants of SSP. SSP solving approaches and discussions can also be found in Tripathy (1992); Sabin and Winter (1986); Feldman and Golumbic (1989).

1.4.4 Practical Cases

Timetabling problems are generally practical cases taken from real-world problems. Each university, school or college has their own constraints and environments that make their problem slightly or heavily different than the ones dealt with in research. At the Université de Technologie de Compiègne (UTC), two timetabling problems are faced each semester: a course and an exam timetabling. The course timetabling is composed into two sub-problems, each processed dif-

ferently. The first sub-problem is building a fixed timetable that constitutes the input for the second sub-problem: the student scheduling problem. The first is done manually by the practitioner by using the data from previous semesters to build a new timetable. The adjustment is done according to expectations on the student choice and the teachers requirements. Once this timetable is stabled, it is made available for students to choose their courses. When the student are finished choosing, the final timetable is made so that all the students are totally assigned to their list of preferences (Vayssade [1978]). The exam timetabling problem is classical with extra constraints which are specific for the UTC (Boufflet and Nègre [1995]). Most constraints can be found in the benchmarks of the literature but some are new and to be found nowhere. We believe that through the UTC problem, we provide a new set of constraints that exist in multiple universities across the world. Some of these constraints involve having multiple sites in which exams are taken. A direct consequence from the first one is having overlapping periods to allow the students move from one site to another. We explain in chapter 4 how the UTC's problem extends an existing benchmark and present a challenging set of instances to be used for method testing.

1.5 Conclusion

In this chapter, we presented a short introduction to combinatorial problems and some of the resolution techniques used in the literature. We also briefly described the timetabling problems discussed in this thesis. The second and the third chapters detail an improved mathematical model and a constraint programming approach for academic exam timetabling. The fourth chapter presents a memetic approach to a practical exam timetabling problem taken from Université de Technologie de Compiègne. The fifth chapter discusses a specific type of course timetabling, the Student Scheduling Problem at Université de Technologie de Compiègne.

CHAPTER 2

AN IMPROVED MODEL FOR EXAM TIMETABLING

Contents

2.1 Introduction

Examination timetabling is one of the most tedious and time-consuming tasks for any academic institution, and which has consequently received considerable

attention over a number of decades. Formally, a set of exams is to be scheduled into a set of timeslots using available rooms while respecting hard and soft constraints. The basic problem generally encountered is a graph coloring problem with extra institutional hard constraints and where penalties incurred for the violation of soft constraints are to be minimized. A solution is said to be feasible if it respects all the hard constraints. Although the problem has broad similarities across different universities, each case is unique in general because institutions have different rules and use different terms to evaluate the quality of solutions. Since a number of approaches have been applied to non-identical problems, clear and meaningful scientific comparisons are difficult. The reader should refer to (Burke et al. [1997b]; Schaerf [1999]; Qu et al. [2009b]) for a detailed overview of examination timetabling.

In the second International Timetabling Competition (ITC2007) (McCollum et al. [2007]; McCollum et al. [2010b]), the examination track introduced a problem encountered within educational institutions across the world. The competition presents significant challenges and plays an important role in bridging the current gap between research and practice by providing real-world datasets for researchers.

The key aim addressed in many papers is achieving high-quality solutions within a short computational time. Data may be preprocessed before use to help reduce the difficulty of the problem (Gogos, Alefragis, and Housos [2012]). As a first step after preprocessing, most approaches aim to find initial feasible solutions. The solutions are later improved by applying heuristics, metaheuristics or hyper-heuristics. Since the competition was introduced, several different approaches have been used. Müller (2009) applied a Hill Climbing and a Great Deluge (Dueck [1993]) with a re-heating strategy. Gogos, Alefragis, and Housos (2012) used Hill Climbing, Simulated Annealing and Integer Programming to ensure a diverse exploration of the search space. A CSP solver, based on Tabu search and iterated local search, was used by Atsuta, Nonobe, and Ibaraki (2008). De Smet (2008) also utilized Tabu Search in conjunction with an open-source business rule management system. A Cell-biology-based approach with Hill Climbing was introduced by Pillay (2008).

Researchers have shown an increasing interest in this real-world problem since the competition was introduced. Turabieh and Abdullah (2011) investigated a

hybrid approach in which a Great Deluge algorithm (Dueck [1993]) is steered using an electromagnetic-like and particle swarm optimization. Hyper-heuristics and adaptive approaches, which showed successful results in previous problems, were also introduced in order to tackle the difficulty of the problem. A hyper-heuristic steers the process of selecting, combining, generating or adapting simpler (meta)heuristics to solve problems efficiently. Adaptive approaches are a learning process in which information is exploited in order to intelligently adapt the behaviour of the algorithm. McCollum et al. (2009) proposed a construction phase followed by Great Deluge. The construction phase is an extension of the work presented by (Burke and Newall [2004]) that is based on the adaption of heuristic orderings. Sabar et al. (2012) derived a hyper-heuristic based on graph coloring constructive ordering heuristics to select exams to be scheduled. Demeester et al. (2012) incorporated the decision part of metaheuristics for accepting/rejecting generated solutions in a hyper-heuristic approach. Soghier and Qu (2013) investigated the impact of bin packing heuristics, and proposed an adaptive hybrid hyper-heuristic approach. Rahman et al. (2014) presented a pure constructive method, based on the squeaky-wheel optimization approach (Clements and Joslin [1999]) and the adaption of heuristics (Burke and Newall [2004]).

The original mathematical model described in (McCollum et al. [2012]) was used for a formal definition of the problem. As McCollum et al. (2012) specified, this model is not optimized for solving the large competition instances, and is subject to further improvements (McCollum et al. [2012]; Burke et al. [2008]). Nevertheless, the model was able to optimally solve the two smallest instances of the Yeditepe University (Parkes and Özcan [2010]). Fonseca and Santos (2013) proposed a modified version of this model.

In practice, solving a real case is an interactive process that may take several days. Because of particular local factors, objectives may be subject to modification by the practitioners, who will often spend a lot of time adjusting the timetable according to the university's needs. Being aware as soon as possible of infeasibilities and hidden constraints is of crucial importance. There are a variety of constraints (such as room capacities) that may make it necessary to place two exams in different periods, apart from the major constraint of students who are required to sit both exams. It can also be useful to provide the practitioners with information about the unavoidable costs or about the existence of feasible solutions that

may totally eliminate certain costs. Such information is needed in practice as an aid to decision making.

The remainder of this chapter is organized as follows. Section 2.2 presents the problem definition and different hard and soft constraints. The original mathematical formulation of hard and soft constraints is given in Section 2.3. Section 2.4 describes our proposed preprocessing that reveals hidden incompatibilities between exams. The improved formulation is described in Section 2.5, in which we present a formulation with a reduced number of hard and soft constraints, along with clique and data-dependent dual-feasible-function valid inequalities. Results, given in Section 2.6, show that the preprocessing reveals a considerable number of hidden constraints and that the formulation proposed gives better results. The details, for each soft constraint, show that the improved formulation yields better solutions, uses less memory and faster run times. Our conclusion is to be found in Section 2.7.

2.2 Problem description

This section provides a brief description of the examination track of the second International Timetabling Competition (ITC2007). The reader should refer to McCollum et al. (2007); McCollum et al. (2010b) for a comprehensive presentation.

The problem introduced in this track involves allocating a set of exams to a set of rooms within an examination session comprising a fixed number of periods, while satisfying a number of hard constraints. A feasible solution is the one in which all hard constraints are respected. The quality of the solution is evaluated using a sum of weighted terms. Each term measures the violation of the respective soft constraint.

The benchmark for this competition track consists of eight public instances and four hidden instances used to counter any over-tuned behaviour in competitors' solvers. Each instance is defined as sets of data, hard constraints, and weighted soft constraints. The day, the start time and the duration of each period are given. A set of rooms, each with a capacity and a weight, is also given. Each exam has an individual duration and a set of enrolled students. Each student has to sit a number of exams. In this track, the hard constraints are as follows:

- A student can sit only one exam at a time.

- An exam cannot be split between rooms.
- An exam cannot be split between periods.
- The duration of an exam allocated to a period must be less than or equal to the duration of the period.
- The capacity of any room is not exceeded at any period.
- Exams can share a room, as long as the capacity of the room is respected.
- Precedences: exam j has to be scheduled prior to exam i.
- Exclusions: exam i must not take place at the same time as exam j.
- Coincidences: two exams i and j must be assigned to the same period.
- Room exclusives: exam i must take place in a room on its own.

Seven soft constraints are used as terms of the evaluation function. For each instance, a set of weights is accordingly provided. The terms are defined in the following way:

- **Two In a Row:** when two exams are allocated back to back on the same day, this term corresponds to the number of students that take these exams multiplied by the weight w^{2R}.
- **Two In a Day:** when two examinations are scheduled not back to back but on the same day where there are three periods or more, this term corresponds to the number of students sitting the two exams multiplied by the weight w^{2D}.
- **Period Spread:** this term corresponds to the sum of occurrences of students who have to sit exams within a fixed period spread defined in the dataset.
- **Front Load:** this term corresponds to the number of large exams scheduled in the latter part of the session multiplied by the weight w^{FL}. The number of periods that constitute the "latter part" and the number of candidates that constitute "large exams" are given.

- **Mixed Duration:** for each period and each room, we look at the number of different exam durations allocated. If all the exams have the same duration we count 0, otherwise we count the number of different durations minus one multiplied by the weight w^{NMD}.
- **Period Penalty:** this term corresponds to the number of exams allocated to a penalized period multiplied by its weight w_p^P.
- **Room Penalty:** this term corresponds to the number of exams allocated to a penalized room multiplied by its weight w_r^R.

The first three terms are an attempt to be as fair as possible to all students taking exams. Since exams having more students enrolled take longer to mark, it is desirable to schedule these exams in the beginning of the examination session (Front Load). The aim of the Mixed-Duration term is to assign exams which are of an equal duration to the same room. Institutions often wish to restrict the use of certain rooms and certain periods to a minimum, and these considerations correspond to the final two terms (Period Penalty and Room Penalty).

2.3 The original model

We present in the sequel the original model proposed by McCollum et al. (2012). As stated by the authors, the aim was to give a clear model whereby one can verify solutions given by the different approaches.

We used the same notation. A brief description of constants, parameters and variables is given. We invite the reader to refer to the original paper for comprehensive details.

2.3.1 The original hard constraints mathematical model

In McCollum et al. (2012), the authors introduced the conflict graph $G(E, A_C)$, where E is the set of exams and an edge $[i, j] \in A_C$ if there is at least one student enrolled in exams i and j. An edge $[i, j]$ is weighted by w_{ij}^C, the number of students taking the two exams. P, R and S denote the sets of periods, rooms and students respectively. For the sake of compactness, the objective function and the hard constraints of the model have been rewritten as:

Minimize:

$$C^{2R} + C^{2D} + C^{PS} + C^{FL} + C^{P} + C^{R} + C^{NMD} \tag{2.1}$$

Subject to

$$\forall i \in E \quad \sum_{p \in P} X^P_{ip} = 1 \tag{2.2}$$

$$\forall i \in E \quad \sum_{r \in R} X^R_{ir} = 1 \tag{2.3}$$

$$\forall i \in E \quad \forall p \in P \quad X^P_{ip} = \sum_{r \in R} X^{PR}_{ipr} \tag{2.4}$$

$$\forall i \in E \quad \forall r \in R \quad X^R_{ir} = \sum_{p \in P} X^{PR}_{ipr} \tag{2.5}$$

$$\forall p \in P \quad \forall r \in R \quad \sum_{i \in E} s^E_i X^{PR}_{ipr} \leq s^R_r \tag{2.6}$$

$$\forall i \in E \quad \forall p \in P \quad d^E_i X^P_{ip} \leq d^P_p \tag{2.7}$$

$$\forall p \in P \quad \forall s \in S \quad \sum_{i \in E} t_{is} X^P_{ip} \leq 1 \tag{2.8}$$

$$\forall (i,j) \in H^{aft} \quad \forall p, q \in P \text{ with } p \leq q \qquad X^P_{ip} + X^P_{jq} \leq 1 \tag{2.9}$$

$$\forall [i,j] \in H^{coin} \quad \forall p \in P \quad X^P_{ip} = X^P_{jp} \tag{2.10}$$

$$\forall [i,j] \in H^{excl} \quad \forall p \in P \quad X^P_{ip} + X^P_{jp} \leq 1 \tag{2.11}$$

$$\left.\begin{array}{r} \forall i \in E^{sole} \quad \forall j \in E \quad i \neq j \quad \forall p \in P \quad \forall r \in R \\ X^P_{ip} + X^R_{ir} + X^P_{jp} + X^R_{jr} \leq 3 \end{array}\right\} \tag{2.12}$$

$$X^P_{ip}, X^R_{ir}, X^{PR}_{ipr} \in \{0,1\} \tag{2.13}$$

The primary boolean decision variables are X^P_{ip} and X^R_{ir}. $X^P_{ip} = 1$ *iff* exam i is

scheduled in period p and $X^R_{ir} = 1$ *iff* exam i is allocated to room r. The secondary boolean variables are: X^{PR}_{ipr}. $X^{PR}_{ipr} = 1$ *iff* exam i is in period p and room r.

Equations (2.2) and (2.3) ensure that all the exams are allocated to a unique period and a unique room. Equations (2.4) and (2.5) link the decision variables. The room capacities are always respected using Equations (2.6) in which s^E_i and s^R_r denote the number of students sitting exam i and the seating capacity of room r respectively. The duration hard constraints are respected using Equations (2.7), in which d^E_i and d^P_p denote the duration of exam i and the duration of period p. The parameters $t_{is} = 1$ *iff* student s is sitting exam i. Equations (2.8) thus enforce the conflict constraints: at any period, any student will be sitting at most one exam.

Sets H^{aft}, H^{coin}, H^{excl} contain the precedence, coincidence and exclusion constraints. Set E^{sole} contains the exams subject to room exclusive constraint. For every pair $(i, j) \in H^{aft}$, exam i must occur strictly after exam j. These precedence contraints are checked by Equations (2.9). For every pair $[i, j] \in H^{coin}$ exams i and j must be allocated to the same period, but not necessarily to the same room. Equations (2.10) ensure that exams i and j are assigned to the same period. The exclusion constraints are enforced by Equations (2.11). Room exclusive constraints are checked using Equations (2.12): for every period, exam $i \in E^{sole}$ is the sole occupier of room r.

All the parameters, s^E_i, s^R_r, d^E_i and d^P_p are embedded in the datasets. The hard constraints for sets H^{aft}, H^{coin}, H^{excl} and E^{sole} are also given.

2.3.2 The original soft constraints mathematical model

The terms of the objective function $C^{2R} + C^{2D} + C^{PS} + C^{FL} + C^P + C^R + C^{NMD}$ (See Equation (2.1)) are assessed by the following:

$$C^{2R} = w^{2R} \sum_{[i,j] \in A_C} w^C_{ij} C^{2R}_{ij} \tag{2.14}$$

$$\left.\begin{array}{r} \forall [i, j] \in A_C \quad \forall p, q \in P \quad \text{with} \quad |p - q| = 1 \\ \text{with} \quad y_{pq} = 1 \quad X^P_{ip} + X^P_{jq} \leq 1 + C^{2R}_{ij} \end{array}\right\} \tag{2.15}$$

$$C^{2D} = w^{2D} \sum_{[i,j] \in A_C} w^C_{ij} C^{2D}_{ij} \tag{2.16}$$

$$\left.\begin{array}{rl} \forall [i,j] \in A_C \quad \forall p,q \in P \quad |q-p| \geq 2 \\ \text{with} \quad y_{pq} = 1 \qquad X^P_{ip} + X^P_{jq} \leq 1 + C^{2D}_{ij} \end{array}\right\} \tag{2.17}$$

$$C^{PS} = \sum_{[i,j] \in A_C} w^C_{ij} C^{PS}_{ij} \tag{2.18}$$

$$\left.\begin{array}{rl} \forall [i,j] \in A_C \quad \forall p,q \in P \quad 1 \leq |q-p| \leq g^{PS} \\ X^P_{ip} + X^P_{jq} \leq 1 + C^{PS}_{ij} \end{array}\right\} \tag{2.19}$$

$$C^{FL} = w^{FL} \sum_{i \in E^{FL}} \sum_{p \geq g^{FL}} X^P_{ip} \tag{2.20}$$

$$C^P = \sum_{i \in E} \sum_{p \in P} w^p X^P_{ip} \tag{2.21}$$

$$C^R = \sum_{i \in E} \sum_{r \in R} w^r X^R_{ir} \tag{2.22}$$

$$\left.\begin{array}{rl} \forall d \in D \quad \forall i \in E^d \quad \forall p \in P \quad \forall r \in R \\ U^D_{dpr} \geq X^P_{ip} + X^R_{ir} - 1 \end{array}\right\} \tag{2.23}$$

$$\forall p \in P \quad \forall r \in R \quad 1 + C^{NMD}_{pr} \geq \sum_{d \in D} U^D_{dpr} \tag{2.24}$$

$$C^{NMD}_{pr} \geq 0 \tag{2.25}$$

$$C^{NMD} = w^{NMD} \sum_{p \in P} \sum_{r \in R} C^{NMD}_{pr} \tag{2.26}$$

$$C^{2R}_{ij}, C^{2D}_{ij}, C^{PS}_{ij}, U^D_{dpr} \in \{0,1\} \tag{2.27}$$

$$C^{2R}, C^{2D}, C^{PS}, C^{FL}, C^P, C^R, C^{NMD}_{pr}, C^{NMD} \in \mathbb{N} \tag{2.28}$$

The boolean variables are C^{2R}_{ij}, C^{2D}_{ij}, C^{PS}_{ij} and U^D_{dpr}. The former three are used to check whether the soft constraints **Two In a Row**, **Two In a Day** and **Period Spread** are violated for an edge $[i,j]$. The latter is set to one *iff* an exam with duration d is assigned in period p and room r. The integer variables C^{2R}, C^{2D},

C^{PS}, C^{FL}, C^{P}, C^{R}, C^{NMD}_{pr} and C^{NMD} are used to compute the terms of the objective function (see Equation (2.1)).

The **Two In a Row** term C^{2R} is set by Equations (2.14) and (2.15), in which the boolean parameter $y_{pq} = 1$ *iff* periods p and q are on the same day.

The **Two In a Day** term C^{2D} is set by Equations (2.16) and (2.17).

The **Period Spread** term C^{PS} is set by Equations (2.18) and (2.19) according to the period spread parameter g^{PS}.

The **Front Load** term is computed using (2.20), where E^{FL} is the set of exams subject to a front load soft constraint, and g^{FL} is the period number from which a front load penalty has to be counted.

The **Period Penalty** and **Room Penalty** terms are taken into account using Equations (2.21) and (2.22).

The **Non Mixed Duration** term is set by Equations (2.23) to (2.26). We denote E^d the set of exams having duration d and D the set of durations of the exams. A U^D_{dpr} decision variable has to be one whenever some exam with duration d uses period p and room r (see Equations (2.23)). Equations (2.24) and (2.25) count the total number of different durations minus one. Hence, the term C^{NMD} is obtained by applying Equation (2.26).

Institutional weights w^{2R}, w^{2D}, w^{FL}, w^{p}, w^{r}, w^{NMD} and the parameter g^{PS} are given. The other ones are easily deduced or computed from the dataset, as for example D, the set of durations, boolean parameters $y_{pq} = 1$ *iff* periods p and q are in the same day, or E^{FL} and g^{FL}, the set of exams subject to front load penalty and the period number from which such a penalty occurs.

We denote $\mathcal{O}$ the original model that consists of Equations (2.1) to (2.28).

2.4 Preprocessing

Gogos, Alefragis, and Housos (2012) proposed a preprocessing step that deduces existing hidden constraints using the existing constraints in the dataset. We propose new preprocessing done in two steps to disclose hidden conflicts. The first exploits capacities of the rooms while the second is based on coincidence constraints so as to propagate conflict constraints. For the sake of clarity, $i \prec j$ represents a precedence constraint between exams i and j and $i \odot j$ a coincidence

Algorithm 1: Preprocessing algorithm

Data: $G(E, A_C)$, E^{coin}, H^{coin}, E^{coin}, E^{aft}, H^{excl}
Result: $G(E, A_{GC})$, $\widetilde{H}^{coin}$, $\widetilde{H}^{aft}$

begin
  $\widetilde{H}^{coin} \leftarrow$ Transitive_closure(H^{coin}) `/* see Section 2.4.1 */`
  $\widetilde{H}^{aft} \leftarrow$ Propagate_precedence_coincidence($\widetilde{H}^{coin}$, H^{aft})
  `/* For each arc` $(i, j) \in \widetilde{H}^{aft}$ `we add the edge` $[i, j]$ `to` A_{GC} `*/`
  $A_{GC} \leftarrow A_C \cup \widetilde{H}^{aft} \cup H^{excl}$
  $A_{GC} \leftarrow$ Room_preprocessing(A_{GC}, $\widetilde{H}^{coin}$)
  `/* U: set of non-propagated edges */`
  $U \leftarrow A_{GC}$
  `/* C: set of propagated edges */`
  $C \leftarrow \emptyset$
  while $(U \neq \emptyset)$ **do**
    Let $a \in U$
    $U \leftarrow U - \{a\}$
    $C \leftarrow C + \{a\}$
    `/* N: set of edges discovered by the propagation */`
    $N \leftarrow$ Propagate_$\mathcal{P}$(a, $\widetilde{H}^{coin}$)
    $A_{GC} \leftarrow A_{GC} \cup N$
    `/* Only new edges are added to` U `*/`
    $U \leftarrow U \cup (N - C)$
  end
end

constraint between exams i and j. Section 2.4.1 presents the previous preprocessing. Our new preprocessing stages are presented in Section 2.4.2.

The preprocessing algorithm (see Algorithm 1) computes $\widetilde{H}^{coin}$ the new set of coincidence constraints, $\widetilde{H}^{aft}$ the new set of precedence constraints and the general conflict graph $G(E, A_{GC})$ where $A_C \subseteq A_{GC}$. The preprocessing is performed in different stages. We apply first the preprocessing stages proposed by Gogos, Alefragis, and Housos (2012) (see Section 2.4.1). Next, we consider the *precedence* and *exclusion* constraints as general conflict constraints. Thereafter, the room preprocessing is applied (see Section 2.4.2). At the last stage, the procedure *Propagate_*$\mathcal{P}$ is repeatedly performed (see Section 2.4.2).

Algorithm 2: Propagate_precedence_coincidence

Data: $\widetilde{H}^{coin}$, H^{aft}
Result: $\widetilde{H}^{aft}$

```
1 begin
      /* For each edge [i,j] ∈ H̃^coin we have two arcs (i,j) and (j,i) */
2     M ← directed_graph(H̃^coin)
3     H̃^aft ← Transitive_closure(M ∪ H^aft)
      /* We remove arcs (i,j) and (j,i) such that [i,j] ∈ H̃^coin */
4     H̃^aft ← H̃^aft − M
5 end
```

2.4.1 Previous preprocessing

In Gogos, Alefragis, and Housos (2012), the authors used a preprocessing stage to discover hidden dependencies between exams. Three types of preprocessing were proposed. The first adjusts the exam durations using the coincidence constraints. Consider a set of exams that should coincide. Since exams must be allocated to the same period, all durations are modified to be equal to the largest one. The second part of the preprocessing applies the *transitive closure* on the precedence (i.e. $\forall\, i,j,k : (i \prec j) \wedge (j \prec k) \Rightarrow (i \prec k)$) and the coincidence constraints (i.e. $\forall\, i,j,k : (i \odot j) \wedge (j \odot k) \Rightarrow (i \odot k)$). The third part is the *propagation of the precedence* using the coincidence. Algorithm 2 performs the last two preprocessing. M is the set of arcs obtained from $[i,j] \in \widetilde{H}^{coin}$.

2.4.2 New preprocessing

A *general conflict constraint* between two exams i and j is defined as a constraint such that exams i and j cannot be scheduled together in any period. We denote this general conflict constraint as $i \parallel j$. As a result, the exclusion and the after constraints represent general conflict constraints since exams involved cannot be placed in the same period.

In $G(E, A_C)$, the weight of an edge $[i,j]$ represents the number of students in common between exams i and j. If $[i,j] \notin A_C$ and $i \parallel j$, we add the edge $[i,j]$ with a null cost since it is a non-student-related conflict constraint (i.e. $w^C_{ij} = 0$).

Room preprocessing

We use the sizes of exams relative to the room capacities to deduce new general conflict constraints. Let i and j be two exams such that $[i,j] \notin A_C$. Let $\mathcal{S}_i$ be the set that contains exam i and all exams k such that $(i \odot k)$. Exams in $\mathcal{S}_i$ have to be scheduled in the same period. Set $\mathcal{S}_j$ is similarly built. The objective is to check whether the exams in $\mathcal{S}_i$ and $\mathcal{S}_j$ can be assigned together to the rooms at any period.

We propose to optimally compute θ_{ij} the maximum number of exams of the set $\mathcal{S}_i \cup \mathcal{S}_j$ that can be allocated together to the same period. If $\theta_{ij} < |\mathcal{S}_i \cup \mathcal{S}_j|$, no exam $l \in \mathcal{S}_i$ can be assigned in the same period with an exam $k \in \mathcal{S}_j$ and conversely. We can therefore add an edge $[l,k]$ for each $l \in \mathcal{S}_i$ and $k \in \mathcal{S}_j$.

We compute θ_{ij} as follows:

maximize:

$$\theta_{ij} = \sum_{r \in R} \sum_{l \in \mathcal{S}_i \cup \mathcal{S}_j} Y_{lr}^R \tag{2.29}$$

subject to:

$$\forall l \in \mathcal{S}_i \cup \mathcal{S}_j \quad \sum_{r \in R} Y_{lr}^R \leq 1 \tag{2.30}$$

$$\forall r \in R \quad \sum_{l \in \mathcal{S}_i \cup \mathcal{S}_j} s_l^E Y_{lr}^R \leq s_r^R \tag{2.31}$$

$$\left.\begin{array}{r} \forall l \in \left(E^{sole} \cap (\mathcal{S}_i \cup \mathcal{S}_j)\right) \quad \forall r \in R \\ \sum\limits_{\substack{m \in \mathcal{S}_i \cup \mathcal{S}_j \\ m \neq l}} Y_{mr}^R + (\mid \mathcal{S}_i \cup \mathcal{S}_j \mid -1) Y_{lr}^R \leq \mid \mathcal{S}_i \cup \mathcal{S}_j \mid -1 \end{array}\right\} \tag{2.32}$$

$$Y_{lr}^R \in \{0,1\} \tag{2.33}$$

Recall that R is the set of rooms and s_r^R denotes the capacity of a room r. The number of students sitting exam l is denoted s_l^E, and Y_{lr}^R is a boolean decision variable set to one *iff* the exam l is allocated to room r.

Equations (2.30) ensure that an exam $l \in \mathcal{S}_i \cup \mathcal{S}_j$ is allocated at most once to a room. Equations (2.31) allow the model to enforce the room capacities. The room exclusive constraints for the exams in $\mathcal{S}_i \cup \mathcal{S}_j$ are checked by Equations (2.32).

Propagation of the general conflict constraints

Let $i \parallel j$ be a general conflict constraint and $[i, j]$ the corresponding edge. Consider that exams j and k are subject to a coincidence constraint $(j \odot k)$ and assume that $[i, k] \notin A_C$. The general conflict constraint $(i \parallel k)$ can be deduced since we have: $(i \parallel j) \wedge (j \odot k) \Rightarrow (i \parallel k)$. Thus a new edge $[i, k]$ can be added with $w^C_{ik} = 0$. We denote $\mathcal{P}$ this propagation that can be repeatedly applied until no new edges can be deduced.

2.5 The improved mathematical model

This section describes the improved formulation that aims to reduce the number of constraints. In the sequel, n^S, n^P, n^E, n^R, and n^D denote the number of students, periods, exams, rooms, and durations respectively. The revisited hard constraints are presented first. Next, we propose a more compact formulation for the Non-Mixed-Duration soft constraint. Finally, we introduce the clique and DDFF valid inqualities.

2.5.1 The revisited constraints

Our objective is to propose modifications that will make it possible to perform computations on the competition instances.

The conflict constraint

Let $\mathcal{N}(i_1, ..., i_k) = \bigcap_{l=1}^{k} \mathcal{N}(i_l)$ where $\mathcal{N}(i)$ corresponds to the usual neighbors of node i in $G(E, A_{GC})$. In McCollum et al. (2012), the authors used $n^S n^P$ equations to ensure that conflict constraints are respected (see Equations (2.8) in Section 2.3.1). We proposed in Arbaoui, Boufflet, and Moukrim (2013) a new formulation for the general conflict constraints so that, for an exam i and a period p, at most $|\mathcal{N}(i)|$ neighbors can be assigned if exam i is not.

We propose to compute exactly α_{ip}, the maximum number of exams of the set $\mathcal{N}(i)$ that can be allocated together in period p. We argue that some exams in $\mathcal{N}(i)$ may not be allocated together to a period. Conflict constraints between the exams in $\mathcal{N}(i)$, or rooms capacities relative to these exam sizes can prohibit

placing all the neighbors in the same period. The other hard constraints (e.g. room exclusive, coincidence, etc.) should also be considered.

Assuming that α_{ip} is known, we propose the $n^E n^P$ improved equations:

$$\forall i \in E \quad \forall p \in P \sum_{j \in \mathcal{N}(i)} X_{jp}^P + \alpha_{ip} X_{ip}^P \leq \alpha_{ip} \tag{2.34}$$

Note that $G(E, A_{GC})$ is used to build the set $\mathcal{N}(i)$. The general conflict constraints are respected, considering an exam i and a period p: at most α_{ip} neighbors of exam i are scheduled if exam i is not. Since $n^E \ll n^S$ (see Table (2.1)), the proposed modification reduces the number of equations. Note that Equations (2.11) are no more needed since exclusion constraints are now embedded in $G(E, A_{GC})$.

For each couple (i, p), the α_{ip} value is computed using the following formulation:

maximize

$$\alpha_{ip} = \sum_{r \in R} \sum_{j \in \mathcal{N}(i)} Z_{jpr}^{PR} \tag{2.35}$$

subject to

$$\forall j \in \mathcal{N}(i) \quad \sum_{r \in R} Z_{jpr}^{PR} \leq 1 \tag{2.36}$$

$$\forall j \in \mathcal{N}(i) \qquad \sum_{r \in R} \sum_{k \in \mathcal{N}(i,j)} Z_{kpr}^{PR} + \mid \mathcal{N}(i,j) \mid \sum_{r \in R} Z_{jpr}^{PR} \leq \mid \mathcal{N}(i,j) \mid \tag{2.37}$$

$$\forall r \in R \quad \sum_{j \in \mathcal{N}(i)} s_j^E Z_{jpr}^{PR} \leq s_r^R \tag{2.38}$$

$$\forall j,k \in \mathcal{N}(i) \quad [j,k] \in H^{coin} \quad \sum_{r \in R} Z_{jpr}^{PR} = \sum_{r \in R} Z_{kpr}^{PR} \tag{2.39}$$

$$\forall j \in \left(E^{sole} \cap \mathcal{N}(i)\right) \quad \forall r \in R \quad \sum_{\substack{k \in \mathcal{N}(i) \\ k \neq j}} Z_{kpr}^{PR} + \mid \mathcal{N}(i) \mid Z_{jpr}^{PR} \leq \mid \mathcal{N}(i) \mid \tag{2.40}$$

$$\forall c \in \mathcal{C}(i) \quad \sum_{r \in R} \sum_{j \in c} Z_{jpr}^{PR} \leq 1 \tag{2.41}$$

The aim of the objective function (2.35) is to maximize the number of exams $j \in \mathcal{N}(i)$ that can be allocated to period p. Z_{jpr}^{PR} are Boolean decision variables, set to one *iff* exam j is allocated to room r at period p. Every exam j is assigned to at most one room using Equations (2.36).

Two exams $j, k \in \mathcal{N}(i)$ can be subject to general conflict constraints. In such a case, if exam j is not scheduled, at most $|\mathcal{N}(i, j)|$ neighbors in the set $\mathcal{N}(i)$ can be allocated to rooms. Hence, we use Equations (2.37) to enforce the general conflict constraints between exams in $\mathcal{N}(i)$. Room capacities are enforced by Equations (2.38).

If a coincidence between $j, k \in \mathcal{N}(i)$ exists, Equations (2.39) ensure that exams j and k are allocated together. The room exclusive constraints are respected using Equations (2.40). For every room, if exam j is subject to a room exclusive constraint, at most $\mid \mathcal{N}(i) \mid$ exams can be scheduled if exam j is not.

Let us now consider the sub-graph induced by $\mathcal{N}(i)$ where only the neighbors of exam i and the associated edges are considered. The set of maximal cliques $\mathcal{C}(i)$ is computed on this induced graph. The clique inequalities, see Equations (2.41), are relative to the sub-graph induced by $\mathcal{N}(i)$ and are useful to speed up the computing of α_{ip}.

Equations (2.35) to (2.41) permit to optimally compute α_{ip} for each couple (i, p).

Hence, in Equations (2.34), we use α_{ip}, the maximum number of neighbors of exam i one can allocate to a period p with respect to the general conflict constraints, room capacities, coincidence constraints and room exclusive constraints.

The room exclusive and precedence constraints

Let $i \in E^{sole}$ be an exam that is subject to a room exclusive constraint, and let p be a period and r a room to which exam i can be allocated. At most n^E exams can be allocated to period p and room r if exam i is not. The equation can therefore be

written as follows:

$$\forall i \in E^{sole} \quad \forall p \in P \quad \forall r \in R \qquad \sum_{\substack{k \in E \\ k \neq i}} X_{kpr}^{PR} \quad + n^E X_{ipr}^{PR} \leq n^E \tag{2.42}$$

There are $\mid E^{sole} \mid n^P n^R$ Equations (2.42) while we have $\mid E^{sole} \mid n^E n^P n^R$ Equations (2.12) in the original model $\mathcal{O}$.

In a similar way, the precedence constraints can be grouped in the following manner:

$$\forall i \in E^{aft} \quad \forall p \in P \sum_{j \in \mathcal{N}_{aft}(i)} \sum_{q \leq p} X_{jq}^P + \mid \mathcal{N}_{aft}(i) \mid X_{ip}^P \leq \mid \mathcal{N}_{aft}(i) \mid \tag{2.43}$$

where E^{aft} denotes the set of exams involved in a precedence constraint. We use the set $\widetilde{H}^{aft}$, resulted from the preprocessing stage (see Section 2.4). We denote $\mathcal{N}_{aft}(i) = \{j \in E^{aft} : i \prec j\}$. There were $\mid H^{aft} \mid n^P n^P$ Equations (2.9), and now we have $\mid E^{aft} \mid n^P$ Equations (2.43).

The coincidence constraints

We define $\mathcal{N}_{coin}(i) = \{j \in E^{coin} : i \odot j\}$, the set of exams that coincide with exam i. We propose the following equations:

$$\forall i \in E^{coin} \quad \forall p \in P \sum_{j \in \mathcal{N}_{coin}(i)} X_{jp}^P = \mid \mathcal{N}_{coin}(i) \mid X_{ip}^P \tag{2.44}$$

There are $\mid E^{coin} \mid n^P$ Equations (2.44) while we have $|H^{coin}| n^P$ Equations (2.10) in the original model $\mathcal{O}$.

The Non-Mixed-Duration constraint

McCollum et al. (2012) used $n^D n^E n^P n^R$ Equations (2.23). We propose the $n^D n^P n^R$ Equations (2.45):

$$\forall d \in D \quad \forall p \in P \quad \forall r \in R \qquad |E^d| U_{dpr}^D \geq \sum_{i \in E^d} X_{ipr}^{PR} \tag{2.45}$$

that ensure that U_{dpr}^D is set to one if there is at least one exam with duration d

assigned to room r at period p, zero otherwise.

2.5.2 Clique and DDFF valid inequalities

The classical clique cuts proved to be efficient with the proposed formulation. We used the following cliques valid inequalities:

$$\forall p \in P \quad \forall c \in \mathcal{C} \quad \sum_{i \in c} X_{ip}^{P} \leq 1 \tag{2.46}$$

where c is a clique, i an exam in clique c, and $\mathcal{C}$ the set of maximal cliques. The clique inequalities are useful for reducing the computation time. We use Östergård (2002) to compute the maximal cliques for each instance.

Carlier, Clautiaux, and Moukrim (2007) introduced a new set of Data-dependent Dual-Feasible Functions (DDFF) to obtain lower bounds for bin packing problems. In the sequel, we define and adapt this DDFF function.

Definition. Let $I = \{1, \cdots, m\}$, $b_1, b_2, \cdots, b_m$ m integer values and B an integer such that $B \geq b_i$ for $i = 1, \cdots, m$. A **DDFF** f associated with B and $b_1, b_2, \cdots, b_m$ is a discrete application from $[0, B]$ to $[0, B']$ such that:

$$\forall I_1 \subset I, \quad \left(\sum_{i \in I_1} b_i \leq B \right) \Rightarrow \left(\sum_{i \in I_1} f(b_i) \leq f(B) \right)$$

It has been proved that any feasible packing for $b_1, b_2, \cdots, b_m$ items in a bin of size B remains feasible for $f(b_1), f(b_2), \cdots, f(b_m)$ items in bin of size $f(B)$. (Carlier, Clautiaux, and Moukrim [2007]) proposed three DDFF functions. We use the function f_1^k that experimentally gives the best results. It is defined for a given parameter k, $1 \leq k \leq \frac{1}{2}B$ and a list of integer values $b_1, b_2, \cdots, b_m$ ($I = \{1, \cdots, m\}$). We consider the set of indices $J = \{i \in I : k \leq b_i \leq \frac{1}{2}B\}$ and $M_B(X, J)$ the optimal value of the one dimensional knapsack problem (1KP) related to J and size X. The value $M_B(X, J)$ is equal to the maximum number of items b_i ($i \in J$) which can be packed in a bin of size X. Assuming that items are sorted by increasing order of size, $M_B(X, J)$ value is obtained in linear time.

Function $f_1^k : [0, B] \rightarrow [0, M_B(B, J)]$ is defined in (Carlier, Clautiaux, and

Moukrim [2007]) as follows:

$$x \to \begin{cases} M_B(B, J) - M_B(B - x, J) & if \ \frac{1}{2}B < x \\ 1 & if \ k \leq x \leq \frac{1}{2}B \\ 0 & otherwise. \end{cases}$$

We use this DDFF as follows. For a given period p, Equations (2.6)

$$\forall p \in P \quad \forall r \in R \quad \sum_{i \in E} s_i^E X_{ipr}^{PR} \leq s_r^R$$

can be viewed as packing items (i.e. exams of size s_i^E) into a bin (i.e. room of size s_r^R). Hence, valid inequalities can be obtained by applying a DDFF function on the sizes of exams and rooms relative to Equations (2.6).

From a practical standpoint, for each Equation (2.6), the function f_1^k is applied on all the s_i^E and on the room capacity s_r^R so as to obtain:

$$\forall p \in P \quad \forall r \in R \quad \sum_{i \in E} f_1^k(s_i^E) X_{ipr}^{PR} \leq f_1^k(s_r^R) \tag{2.47}$$

We denote $\mathcal{M}$ the proposed model that consists of Equations (2.1) to (2.6), (2.13) to (2.22), (2.24) to (2.28), (2.34), (2.42) to (2.47).

2.6 Experimental results

Tests were done using CPLEX 12.5 IBM (2012) MIP solver with a single thread and *MipEmphasis* parameter set to *feasibility*, gcc version 4.4.7, on a machine with an Intel Xeon E5-2670 and 8 GB of RAM. We set a one-hour time limit for models $\mathcal{O}$ and $\mathcal{M}$. The computing times are reported in seconds.

2.6.1 ITC2007 instances

Characteristics of the ITC2007 datasets are given in Table 2.1. We report in the first column the instance label, and, the next four columns show number of exams, number of students, number of periods and number of rooms. Column $|H^{aft}|$ denotes the number of precedence constraints, $|H^{excl}|$ the number of exclusion

Table 2.1 – Characteristics of the ITC2007

	n^E	n^S	n^P	n^R	$\|H^{aft}\|$	$\|H^{excl}\|$	$\|H^{coin}\|$	$\|E^{sole}\|$
1	607	7891	54	7	9	1 (1)	2	0
2	870	12743	40	49	3	1 (1)	4	2
3	934	16439	36	48	1	1 (0)	82	15
4	273	5045	21	1	0	16 (5)	4	0
5	1018	9253	42	3	6	5 (5)	16	0
6	242	7909	16	8	2	2 (2)	18	0
7	1096	14676	80	15	6	9 (9)	13	0
8	598	7718	80	8	15	0	5	1
9	169	655	25	3	7	1 (1)	2	0
10	214	1577	32	48	9	0	49	0
11	934	16439	26	40	1	1 (0)	82	15
12	78	1653	12	50	0	7 (6)	2	7

Table 2.2 – Results of the preprocessing stages for ITC2007 instances

	$\|\widetilde{H}^{aft}\|$	$\|\widetilde{H}^{coin}\|$	$\mathcal{R}$	$\mathcal{P}$	$\|A_C\|$	$\|A_{GC}\|$	%	t_{pre}
1	10 (0)	2	928	92	9287	10308	10.9%	30
2	3 (3)	4	0	41	4421	4466	1.0%	28
3	1 (0)	82	0	2477	11410	13887	21.7%	97
4	0	4	188	31	5568	5792	4.0%	1
5	7 (7)	19	0	378	4500	4890	8.6%	1
6	3 (3)	18	4	489	1795	2293	27.7%	3
7	6 (6)	13	1	491	11595	12102	4.3%	1
8	16 (1)	5	902	190	8120	9213	13.4%	43
9	8 (7)	2	49	23	1113	1193	8.0%	2
10	18 (8)	53	1	1198	1133	2340	106.5%	1
11	1 (0)	82	0	2477	11410	13887	21.7%	104
12	0	2	18	57	554	635	14.6%	1

constraints, $|H^{coin}|$ the number of coincidence constraints, and $|E^{sole}|$ the number of exams subject to room exclusive constraints.

Note that column $|H^{excl}|$ reports first the initial number of exclusion constraints, while in parenthesis, we show the number of edges corresponding to these general conflict constraints that are not embedded in $G(E, A_C)$.

Results of preprocessing on ITC2007 instances

The preprocessing results are reported in Table 2.2. Column $|\widetilde{H}^{aft}|$ shows the new number of precedence constraints, and, $|\widetilde{H}^{coin}|$ the new number of coincidence constraints. The number of new general conflict constraints deduced when ap-

plying the room preprocessing stage is reported in column $\mathcal{R}$, and, the number of new general conflict constraints deduced when applying *Propagate_*$\mathcal{P}$ is in column $\mathcal{P}$. Column $|A_C|$ reports the initial number of edges in $G(E, A_C)$. The total amount of edges after the preprocessing can be shown in column $|A_{GC}|$, and, column % corresponds to the percentage of new edges added. Column t_{pre} reports the computing times for the preprocessing.

By comparing columns $|H^{aft}|$ in Table 2.1 and $|\widetilde{H}^{aft}|$ in Table 2.2, we show that new precedence constraints are discovered for instances 1, 5, 6, 8, 9 and 10. For example, there are nine precedence constraints in the initial data of instance 10 (see Table 2.1) and nine new precedence constraints are discovered as it can be seen in Table 2.2. The number of new edges that can be added to $G(E, A_{GC})$ is reported in parenthesis in column $|\widetilde{H}^{aft}|$. For instance 10, there are only eight new edges. By comparing column $|\widetilde{H}^{coin}|$ (see Table 2.2) and $|H^{coin}|$ (see Table 2.1), we see that new coincidence constraints can be exhibited for instances 5 and 10.

The room preprocessing is useful to deduce a significant number of general conflict constraints for instances 1, 4 and 8 (see column $\mathcal{R}$). A large number of general conflict constraints are also deduced using propagation *Propagate_*$\mathcal{P}$. The preprocessing has a significant impact, the number of edges increases up to 106.5% (see column % in Table 2.2). Column t_{pre} shows that the preprocessing is done in a short time.

Table 2.3 shows the impact of the preprocessing on the maximum clique size. The maximum cliques have been computed using the code described in (Östergård [2002]). Columns ω_{A_C} and $n_{\omega_{A_C}}$ ($\omega_{A_{GC}}$ and $n_{\omega_{A_{GC}}}$ respectively) give the size of the maximum clique and the number of different maximum cliques in $G(E, A_C)$ ($G(E, A_{GC})$ respectively). Note that the size of the maximum clique has more than doubled for instances 1 and 8. Note also that the number of maximum cliques increases for instances 6, 7, 10 and 12.

The α_{ip} values are needed for model $\mathcal{M}$ and are computed using the MIP detailed in Section 2.5.1. The computing times for all the α_{ip} values are displayed in column $t_{\alpha_{ip}}$ in Table 2.3. Note that less than half an hour is needed for each instance.

Table 2.3 – Impact of the preprocessing stage on maximum cliques and computing time for the α_{ip} value for ITC2007 instances

	ω_{A_C}	$n_{\omega_{A_C}}$	$\omega_{A_{GC}}$	$n_{\omega_{A_{GC}}}$	$t_{\alpha_{ip}}$
1	20	4	**49**	4	172
2	15	1	15	1	124
3	21	8	21	8	1150
4	17	6	**18**	6	92
5	13	2	13	2	43
6	13	5	13	54	52
7	16	66	16	70	213
8	17	49	**48**	2	330
9	10	6	**11**	5	6
10	18	3	18	2688	412
11	21	8	21	8	1059
12	12	4	12	7	6

Results for model $\mathcal{O}$ and model $\mathcal{M}$ on ITC2007 instances

Table 2.4 reports results for tests conducted on the ITC2007 datasets using models $\mathcal{O}$ and $\mathcal{M}$. The instance label is reported in the first column. For each model, we consider every soft constraint separately so as to obtain upper bounds or optimal values. The time limit was set to one hour. Finally, all the soft constraints are considered together in the objective function. For each case, the Upper Bound value (UB) and computing time (t) in seconds are provided. For every soft constraint, the results assessed using models $\mathcal{O}$ and $\mathcal{M}$ are tabulated. In Table 2.4, " - " stands for *time limit reached*, **NS** for *no solution found* and **OM** for *out of memory*.

Table 2.4 is divided in two parts, according to the number of soft constraints generated in model $\mathcal{M}$. The upper part of Table 2.4 contains the results for the Front-Load, Period, Room and Non-Mixed-Duration penalties. We call these soft constraints *linear soft constraints* because, when they are considered, the number of constraints generated is in the order of $O(n^E)$. The lower part contains the Two-In-a-Row, Two-In-a-Day, and Period-Spread penalties. When these soft constraints are taken into account, the number of constraints in model $\mathcal{M}$ is in the order of $O(n^E n^E)$.

For the Front-Load penalty, model $\mathcal{M}$ finds two more optimal solutions (i.e. instances 1 and 8) and better feasible solutions for three instances relative to model $\mathcal{O}$ (i.e. instances 2, 3 and 4). For the Period penalty, model $\mathcal{M}$ reaches optimality for ten out of twelve instances and gives a feasible solution for in-

Table 2.4 – Results for model $\mathcal{O}$ and model $\mathcal{M}$ on ITC2007 instances

I	Front Load				Period				Room				Non Mixed Duration			
	$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$	
	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t
1	NS	-	125	1304	NS	-	0	202	NS	-	NS	-	NS	-	0	1281
2		OM	15	-		OM	0	1071		OM	0	53		OM	0	227
3		OM	440	-		OM	0	326		OM	0	398		OM		OM
4	45	-	40	-	NS	-	250	-	0	647	0	49	0	643	0	62
5	50	22	50	48	0	7	0	12	0	4	0	7	0	5	0	9
6	375	13	375	15	65	-	75	-	950	77	950	59	25	-	25	-
7	0	196	0	171	0	143	0	153	0	206	0	186	0	644	0	431
8	520	-	0	97	0	89	0	51	2770	-	0	80	0	197	0	174
9	0	6	0	5	0	1	0	0	0	0	0	0	0	4	0	4
10	0	14	0	20	0	3	0	5	0	18	0	19	0	16	0	21
11		OM	NS	-		OM	0	2090		OM	0	1856		OM	NS	-
12	40	99	40	20	0	7	0	5	0	21	0	4	0	25	0	7

I	Two In a Row				Two In a Day				Period Spread				All the soft constraints			
	$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$		$\mathcal{O}$		$\mathcal{M}$	
	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t	UB	t
1	NS	-	NS	-	0	275	0	317		OM		OM		OM		OM
2		OM	0	181		OM	0	136		OM	0	182		OM	NS	-
3		OM		OM		OM		OM		OM		OM		OM		OM
4	NS	-	NS	-	NS	-	2135	-	NS	-	NS	-	NS	-	NS	-
5	0	20	0	25	0	13	0	21	7145	-	4302	-	NS	-	123005	-
6	5460	-	5160	-	0	11	0	9	NS	-	NS	-	NS	-	NS	-
7		OM	0	687	0	56	0	47		OM		OM		OM		OM
8	0	231	0	296	0	26	0	16		OM		OM		OM		OM
9	0	21	0	17	0	0	0	0	1054	-	1525	-	6559	-	5860	-
10	0	23	0	26	0	5	0	5	13097	-	12814	-	39250	-	20195	-
11		OM	NS	-		OM	NS	-		OM		OM		OM		OM
12	4725	-	3675	-	0	9	0	4	2275	-	2260	-	7027	-	6121	-

stance 4, whereas model $\mathcal{O}$ succeeded for six out of twelve and could not solve instance 4. Model $\mathcal{O}$, however, is better in the case of instance 6. For the Room penalty, model $\mathcal{M}$ finds four new optimal solutions compared to model $\mathcal{O}$ (i.e. instances 2, 3, 8 and 11). For the Non-Mixed-Duration penalty model $\mathcal{O}$ reached optimality for seven out of twelve instances, while the proposed model gains two instances (i.e. instances 1 and 2).

The model $\mathcal{O}$ cannot run instances 2, 3 and 11 regardless the soft constraint considered due to an out of memory (OM). This is due to the large number of conflict constraints based on students in model $\mathcal{O}$. Despite the possibility of running it, instance 11 remains difficult for the two models. Instance 1 represents another challenging dataset. Model $\mathcal{O}$ cannot find any solution for the four linear soft constraints considered within the one hour time limit, whereas model $\mathcal{M}$ succeeded in optimally solve three of them.

The Two-In-a-Row, Two-In-a-Day, and Period-Spread soft constraints are difficult to deal with. The two models have difficulties to find feasible solutions within the one-hour time limit, especially for Period-Spread penalty. For the Two-In-a-Row penalty, model $\mathcal{O}$ reaches optimality for four instances whilst model $\mathcal{M}$ finds two additional optimal solutions (i.e. instances 2 and 7). For the Two-In-a-Day penalty, both models find an optimal solution for instances 1, 6, 7, 9 and 12 because, in these instances, days have only one or two periods, which obviously leads to a null cost notwithstanding the solution obtained. One can also observe that one additional optimal solution (instance 2), and an upper bound are attainable (instance 4) when applying model $\mathcal{M}$ on the Two-In-a-Day soft penalty. We notice that the Period-Spread soft constraint causes an out of memory for five out twelve instances for model $\mathcal{M}$ and six out twelve instances for model $\mathcal{O}$. Despite the large number of constraints, model $\mathcal{M}$ is able to provide the optimal solution for instance 2 within the one hour time limit.

When all the soft constraints are considered at once, model $\mathcal{M}$ find better solutions than model $\mathcal{O}$ and a feasible solution for instance 5. We note that both models struggle in most cases due to the out of memory though. Considering all the soft constraints turned finding feasible solutions into a complicated task for both models.

We report a synthesis of the 96 runs done on ITC2007 instances in Table 2.5. The first two columns report the number of the out of memory (OM) that oc-

Table 2.5 – Synthesis for the 96 runs on ITC2007 instances

OM count		ranking $\mathcal{O}$ and $\mathcal{M}$ when solutions are achieved		
$\mathcal{O}$	$\mathcal{M}$	($\mathcal{O}$ = $\mathcal{M}$)	($\mathcal{O}$ better_than $\mathcal{M}$)	($\mathcal{O}$ worse_than $\mathcal{M}$)
31	13	39	2	30

cured over the 96 runs. The comparison between the two columns shows that compact model $\mathcal{M}$ fits better within the memory limit. Columns ($\mathcal{O}$ = $\mathcal{M}$), ($\mathcal{O}$ better_than $\mathcal{M}$), and ($\mathcal{O}$ worse_than $\mathcal{M}$) report the number of equal solutions achieved when computations are feasible for both models, the number of better solutions achieved by model $\mathcal{O}$, and the number of better solutions achieved by model $\mathcal{M}$, respectively. In 25 cases, $\mathcal{O}$ and $\mathcal{M}$ cannot obtain a feasible solution, either the models cannot fit into memory (OM) or the models cannot find a feasible solution within the time limit (NS). In the remaining cases, it can be seen that model $\mathcal{M}$ provides better results for the majority of cases.

The results in Table 2.4 also show that model $\mathcal{M}$ succeeded in solving to optimality thirteen cases more than the original model $\mathcal{O}$. In most cases, the computing times of model $\mathcal{M}$ are shorter than the ones of model $\mathcal{O}$. The proposed model $\mathcal{M}$ obtains better results within 8 GB of RAM and one hour time limit.

2.6.2 Yeditepe instances

Characteristics of the Yeditepe datasets are given in Table 2.6. We report in the first column the instance label, and, the next four columns show number of exams, number of students, number of periods and number of rooms. Columns ω_{A_C} and $n_{\omega_{A_C}}$ give the size of the maximum clique and the number of different maximum cliques in $G(E, A_C)$. The computing times for all the α_{ip} values are displayed in column $t_{\alpha_{ip}}$.

Results of preprocessing on Yeditepe instances

No precedence, coincidence, exclusion or room exclusive constraints are embedded in the Yeditepe instances (Parkes and Özcan [2010]). Moreover, the capacities

Table 2.6 – Characteristics of the Yeditepe datasets

	n^E	n^S	n^P	n^R	ω_{A_C}	$n_{\omega_{A_C}}$	$t_{\alpha_{ip}}$
yue20011	126	559	18	2	14	78	1
yue20012	141	591	18	2	17	8	1
yue20013	26	234	6	2	6	4	1
yue20021	162	826	21	2	16	17	1
yue20022	182	869	21	2	20	4	1
yue20023	38	420	6	1	6	2	1
yue20031	174	1125	18	2	14	101	1
yue20032	210	1185	18	2	16	14	1

Table 2.7 – Results for model $\mathcal{O}$ and model $\mathcal{M}$ on Yeditepe instances

I	$\mathcal{O}$		$\mathcal{M}$	
	UB	t	UB	t
yue20011	147	-	103	-
yue20012	135	-	279	-
yue20013	29	3	29	2
yue20021	113	-	107	-
yue20022	434	-	388	-
yue20023	56	38	56	16
yue20031	287	-	361	-
yue20032	8342	-	645	-

of the rooms are large enough so that no new general conflict constraints can be deduced using the room preprocessing. As expected, the preprocessing is unable to exhibit new edges for these instances.

Results for model $\mathcal{O}$ and model $\mathcal{M}$ on Yeditepe instances

Table 2.7 reports the results obtained on the Yeditepe instances. The instances contain only Two-In-a-Row and Room penalty soft constraints. A deep look at this dataset shows that Room penalty can be avoided by both models. For all Yeditepe instances but one, a very large room with a high cost has been added to fit into the ITC2007 problem formulation. We experimentally observed that Room penalty soft constraint has no impact on the costs of solutions. For the sake of compactness, we only report in Table 2.7 experiments conducted when considering the two soft constraints.

The two smallest instances *yue*20013 and *yue*20023 are optimally solved by

both models.

Table 2.7 shows that model $\mathcal{M}$ finds four better solutions, while $\mathcal{O}$ finds two better solutions.

2.7 Conclusion

Preprocessing stages and an improved mathematical model were presented for examination timetabling problems that fall within the scope described in the second International Timetabling Competition (ITC 2007). The preprocessing stages reveal general conflict constraints between exams even if they have no students in common. The original model and the improved model were compared using the ITC 2007 and Yeditepe datasets within a one-hour time limit.

The improved model is more compact and the number of hard and soft constraints was significantly reduced. The original model cannot fit into the memory in 31 cases whereas the proposed model cannot fit only in 13 cases. The preprocessing stages helped find larger cliques used by the improved model as valid inequalities. We introduced Data-dependent Dual Feasible Function valid inequalities that proved to be efficient. When both models can be loaded, the improved model has better results on the majority of instances.

CHAPTER 3

LOWER BOUNDS AND CP TECHNIQUES FOR EXAM TIMETABLING

Contents

3.1 Introduction

Exam timetable problems are hard problems to solve. Heuristics and metaheuristics are able to build good solution that provide upper bounds. Today, exact meth-

ods based on, for instance, MIP formulations cannot reach optimality except for small problems. Lower bounds are useful for evaluate how far from the optimal solution are the practical ones.

We consider the ITC2007 examination timetabling problem discussed in Chapter 2. We invite the reader to refer to Section 2.2 for a description of the problem.

We propose lower bounds for Two-In-a-Row, Two-In-a-Day, and Period-Spread penalties. As it has been presented in Chapter 2, these penalties are still difficult to deal with even though improvements have been proposed and tested.

As it can be noticed in the first model presented in McCollum et al. (2012), it is probably because they are intrinsically quadratic. The authors proposed a linearization which results a quadratic number of constraints. We propose another linearization to reduce the number of constraints. We observed that current MIP solvers encounter difficulty running the models developed so far. We propose a constraint programming formulation to test another exact approach based on this paradigm. We test, under this context, the new linearization.

3.2 Lower Bounds

The aim behind the spacing soft constraints Two In a Row, Two In a Day and Period Spread is to place exams for the same student as far apart as possible. For most solutions the major part of the penalty imposed is due to these criteria.

3.2.1 Clique Size Limits for Unavoidable Penalties

Based on the size of the clique, we determine the unavoidable cost imposed by the exams of these cliques. This technique is applied on three soft constraints: *Two In a Row*, *Two In a Day* and *Period Spread*, denoted ***spacing soft constraints***. For the remaining soft constraints, we use the formulation presented in the previous chapter in section 2.5 to solve them. For each soft constraint, the objective function to be minimized is the corresponding penalty and the constraints are all the hard constraints of the problem. To the best of our knowledge, there exists no lower-bound-related work on the ITC2007 problem.

To assess lower bounds for these soft constraints, we propose using a set of cliques $\mathcal{C}$ selected according to their sizes. We denote k the size of a clique c. A

day is said to be of type Di if it has i periods, and n^{Di} is the number of days of type Di. Not all the possible Di exist for a particular instance (e.g. instance 2 has $D2$, $D3$ and $D4$ types of day but no $D1$). We denote δ the set of numbers of periods that corresponds to the types of day of an instance. As an example, for instance 2, we have $\delta = \{2, 3, 4\}$ since we have $D2$, $D3$ and $D4$ types of days.

We establish the following limits:

Proposition 3.1 *If $k > L^{2R} = \sum_{i \in \delta} \lceil \frac{i}{2} \rceil n^{Di}$, then we have at least one Two-In-a-Row penalty.*

exams of a clique c are pairwise adjacent. For $D1$ and $D2$ types of day, one exam of a clique can be allocated without any two-in-a-row conflict. For $D3$ type of day, two exams of a clique can be allocated without any two-in-a-row conflict. Hence, for Di type of day i/2 exams of a clique can be allocated without any two-in-a-row conflict. So, the limit L2r holds. Assume now a clique c such $k > L^{2R}$, we have a least a two-in-a-row penalty.

Proposition 3.2 *If $k > L^{2D} = n^{D1} + 2\sum_{i \in (\delta \setminus \{1\})}^{4} n^{Di}$, then we have at least one Two-In-a-Day penalty.*

For the two-in-a-row conflict, one exam can be placed in a $D1$ day and two exams can be placed for $D2$, $D3$ and $D4$ days. Hence, for a clique c of size $k > L^{2D}$, we have at least a two-in-a-day penalty.

Proposition 3.3 *If $k > L^{2RD} = \sum_{i \in \delta} n^{Di}$, then there is at least one penalty due either to Two In a Row, or to Two In a Day.*

To avoid having both two-in-a-row and two-in-a-day penalties, we can schedule at most one exam per day. Hence, a clique c of size $k > L^{2RD}$, we have at least a two-in-a-row or two-in-a-day penalty.

Proposition 3.4 *If $k > L^{PS} = \left\lceil \frac{n^P}{g^{PS}+1} \right\rceil$, then we have at least one Period-Spread penalty.*

For a clique c of size k, a minimum gap of g^{PS} periods is required to avoid having a period-spread penalty. Hence, if $k > L^{PS}$, we have at least a period-spread penalty.

	n^P	g^{PS}	L^{2R}	L^{2D}	L^{2RD}	L^{PS}
1	54	5	29	ns	29	9
2	40	1	24	26	13	20
3	36	4	24	24	12	8
4	21	2	14	14	7	7
5	42	5	28	28	14	7
6	16	20	8	ns	8	1
7	80	10	40	ns	40	8
8	80	15	41	79	41	5
9	25	5	13	ns	13	5
10	32	20	22	22	12	2
11	26	4	17	18	9	7
12	12	5	7	ns	12	3

Table 3.1 – Limits on sizes of cliques beyond which a penalty applies

All these limits can be stated by counting the number of exams that can be scheduled without the considered soft constraint. Table 3.1 reports the values for the limits on the size of a clique beyond which a violation of the corresponding soft constraint would necessarily occur. When we have only $D1$ or $D2$ day types, there are no Two-In-a-Day penalties (**ns** in Table 3.1).

A set of cliques for each instance is built using the approach proposed by Östergård (2002). Each clique c is evaluated as described in Sections 3.2.2 and 3.2.3.

3.2.2 Two In a Row and Two In a Day Penalties

We now focus on the C^{2R} and C^{2D} lower bound computation. We have $L^{2R} \geq L^{2RD}$, and also $L^{2D} \geq L^{2RD}$ (except when there are no Two-In-a-Day penalties). We therefore consider L^{2RD}: the number of cliques involved in the computation should be potentially larger, as shown by a comparison of the columns L^{2R}, L^{2D} and L^{2RD} in Table 3.1. The objective function for the modified model $\mathcal{M}$ (see Section 2.5) is thus to minimize $(C_b^{2R} + C_b^{2D})$ (subscript b means "both"), and the other soft constraints are not considered. As a result, we determine the optimal cost for each clique.

Unfortunately, we observe in practice that the computation time for a clique

is of the order of hours even when we use the modified model $\mathcal{M}$, and there are a large number of cliques to evaluate.

We propose a new model that is more effective for computing Two-In-a-Row and Two-In-a-Day penalties per clique. Since we are considering a clique $c \in \mathcal{C}$ of size k, exams are pairwise adjacent: they have to be assigned to k different periods. The examination session extending over n^P periods corresponds to different day types. There are n^{Di} days of type Di with i periods.

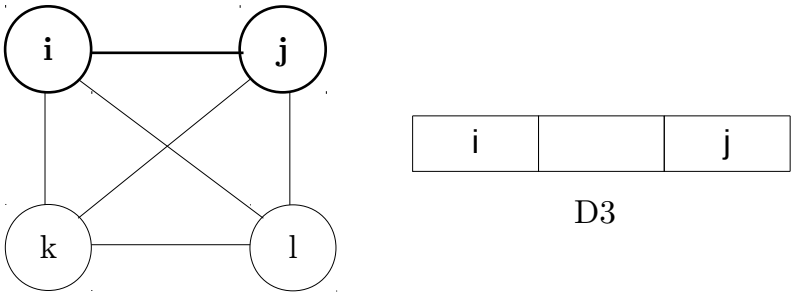


Figure 3.1 – Clique configuration on a D3 day type

The idea is to start by building permutations of the exams in a clique c for each day type comprising the instance ($D1$, $D2$, $D3$, $D4$), and then to optimally select those that can cover all the exams of c with the minimum Two-In-a-Row and Two-In-a-Day cost. There is a large number of permutations. Fortunately, not all the permutations need to be used to find the optimal solution for a clique c.

To provide the reader with some insights on how the permutations are selected, let us consider a $D3$ day and a 4-exam clique and focus on two exams $i, j \in c$ (see Figure 3.1). There are many feasible permutations of these two exams that can be allocated to the three periods of a $D3$. The periods inside a $D3$ day in which the two exams are scheduled do not matter: only the penalty is important. Hence, for a couple of exams i and j a unique permutation with the minimum cost has to be considered. This rationale can be applied on the other types of days.

We evaluate the contribution for a clique c using the following model:

minimize:

$$\sum_{\sigma \in \Pi} C_\sigma X_\sigma \tag{3.1}$$

subject to:

$$\forall j \in [1,k] \qquad \sum_{\sigma \in \Pi} a_{j\sigma} X_\sigma = 1 \tag{3.2}$$

$$\forall i \in \delta \quad \sum_{\sigma \in \Pi_i} X_\sigma \leq n^{Di} \tag{3.3}$$

$$X_\sigma \in \{0,1\} \tag{3.4}$$

Subscript σ denotes a permutation. X_σ is a Boolean decision variable, equal to 1 if permutation σ is used, 0 otherwise. The permutations are grouped by days, i.e. Π_i is the set of permutations for the days with i periods. Each permutation σ has a cost denoted C_σ that corresponds to its Two-In-a-Row and Two-In-a-Day penalty. Exams j of a k-clique are here numbered from 1 to k. The parameter $a_{j\sigma} = 1$ if exam j belongs to permutation σ, 0 otherwise.

Equations (3.2) ensure that each exam is assigned exactly once (3.2). The usage of the different days is enforced by (3.3). We use at most the associated number of days for each type. Note that the days are anonymous: the model allocates the permutations to an abstract "day", independently of the actual day to which this might be made to correspond in practice. We denote $\mathcal{K}$ the proposed model.

At this stage, the optimal contributions are computed for each clique c in the set $\mathcal{C}$ for which $k = \mid c \mid > L^{2RD}$.

3.2.3 Period Spead Penalty

In practice, we observe that the time spent finding an optimal solution for a clique c for C^{PS} is longer than the time spent evaluating $(C_b^{2R} + C_b^{2D})$ when we use the modified model $\mathcal{M}$ (see Section 2.5). Moreover, the total number of cliques whose cardinality exceeds L^{PS} is huge. We propose the following scheme for evaluating the contributions of a clique c for the C^{PS} penalty.

We first recall Equations (2.18) and (2.19) that are used to compute the penalty of each clique:

$$C^{PS} = \sum_{[i,j] \in A_C} w_{ij}^C C_{ij}^{PS}$$

$$\left.\begin{array}{c} \forall [i,j] \in A_C \quad \forall p,q \in P \quad 1 \leq |q-p| \leq g^{PS} \\ X^P_{ip} + X^P_{jq} \leq 1 + C^{PS}_{ij} \end{array}\right\}$$

Assume a clique $\mid c \mid = k > L^{PS}$: there is at least one Period-Spread penalty. First we determine α_k, the minimum number of edges that contribute to the penalty of any clique c of size k. Each edge $[i,j]$ represents a term $w^C_{ij} C^{PS}_{ij}$ in the objective function The minimum number of edges α_k for this size of clique k is consequently known, irrespective of clique c. Next, to evaluate clique c, we take the α_k edges that have the smallest penalties.

To compute α_k, the idea is to use the proposed model to find an optimal spacing between k exams over the n^P periods that formally corresponds to an optimal pattern (exams allocated to periods). It is important to remark that for given n^P and g^{PS}, each value α_k is the same for all the cliques such that $\mid c \mid = k$. Hence, α_k has to be computed once for all the cliques of size k.

We consider formally a clique of size k with a set of exams $\{i_1, i_2, \cdots, i_k\}$. All the w_{ij} are set to 1. Let us now assume the optimal spacing between the k exams with the minimum value α_k. Since exams are anonymous and $w_{ij} = 1$, two exams can swap places. As a consequence, there are $k!$ feasible permutations. Without lack of generality, we introduce the following total order: $(i_1 \prec i_2 \prec \cdots \prec i_k)$. This total ordering is useful to speed up the computation. Then we use the modified model $\mathcal{M}$ to minimize C^{PS} for this optimal spacing and the other soft constraints are not considered. Unfortunately it is still time consuming to prove optimality for certain cliques. We stop the computation after ten minutes for each clique. When the time limit is reached, we get a lower bound for α_k.

Before computing the penalties, we compute α_k *a priori* for each different clique size such that $k > L^{PS}$. For each clique c, the proposed evaluation that counts the α_k smallest weights of the edges of clique c does not represent the optimal value for clique c, but it gives a lower bound for its C^{PS} criterion.

3.2.4 Evaluation of A Set of Cliques With Unavoidable Penalties

Let c and c' two cliques and $[i,j] \in c, c'$. The penalties of c and c' cannot be summed since we have no guarantee that the edge $[i,j]$ does not contribute in the penalties of both cliques.

Assume we have a set of individual cliques, each clique has at least an unavoidable penalty and each clique has been assessed as explained.

Proposition 3.5 *Let $\mathcal{F}$ be a family of cliques embedded in $G(E, A_{GC})$ such that: $\forall c, c' \in \mathcal{F}$ $|c \cap c'| \leq 1$, therefore $\sum_{c \in \mathcal{F}} C^{2R}(c) + C^{2D}(c) + C^{PS}(c)$ is a lower bound.*

$\mathcal{F}$ is a family of edge-disjoint cliques. Since no edge is shared between two cliques in the family, the penalties of these cliques can be summed.

Such a family $\mathcal{F}$ as presented in the above proposition is built using a greedy algorithm. In Section 3.2.2

3.2.5 Remaining Penalties

To compute lower bounds for the remaining penalties (frontload, room, period and non mixed duration), we use the model $\mathcal{M}$ with the preprocessing and the valid inequalities, described in Section 2.5. We run model $\mathcal{M}$ on each each soft constraint individually. The objective function is composed of one term, corresponding to the penalty of the soft constraint. The aim of running model $\mathcal{M}$ is to see whether it is able to reach optimality for the four soft constraints. If it is possible, this would suggest that a big part of the difficulty of the instances comes from the spacing constraints Two In a Row, Two In a Day and Period Spread.

3.3 A Modified MIP for Exam Timetabling

In the previous chapter, Section 2.5 presented the set of the different hard and soft constraints. The spacing soft constraints, originally non-linear, were linearized to be used in the current MIP.

This linearization, however, presents a drawback. It generates a quadratic number of variables and constraints (in the order of n^{E^2} variables and $n^p * n^{E^2}$ constraints). This led to an overload in the memory required by the solver to solve the MIP. We propose a new linearization based on the work of Glover (1975) in which the linearization requires the order of n^E variables and $n^P * n^E$ constraints.

We recall the linearization presented in the previous chapter for the spacing

constraints:

$$C^{2R} = w^{2R} \sum_{[i,j] \in A_C} w_{ij}^C C_{ij}^{2R} \tag{3.5}$$

$$\left.\begin{array}{r} \forall [i,j] \in A_C \quad \forall p,q \in P \quad \text{with} \quad |p-q| = 1 \\ \text{with} \quad y_{pq} = 1 \quad X_{ip}^P + X_{jq}^P \leq 1 + C_{ij}^{2R} \end{array}\right\} \tag{3.6}$$

$$C^{2D} = w^{2D} \sum_{[i,j] \in A_C} w_{ij}^C C_{ij}^{2D} \tag{3.7}$$

$$\left.\begin{array}{r} \forall [i,j] \in A_C \quad \forall p,q \in P \quad |q-p| \geq 2 \\ \text{with} \quad y_{pq} = 1 \qquad X_{ip}^P + X_{jq}^P \leq 1 + C_{ij}^{2D} \end{array}\right\} \tag{3.8}$$

$$C^{PS} = \sum_{[i,j] \in A_C} w_{ij}^C C_{ij}^{PS} \tag{3.9}$$

$$\left.\begin{array}{r} \forall [i,j] \in A_C \quad \forall p,q \in P \quad 1 \leq |q-p| \leq g^{PS} \\ X_{ip}^P + X_{jq}^P \leq 1 + C_{ij}^{PS} \end{array}\right\} \tag{3.10}$$

$$C_{ij}^{2R}, C_{ij}^{2D}, C_{ij}^{PS} \in \{0,1\} \quad C^{2R}, C^{2D}, C^{PS} \in \mathbb{N} \tag{3.11}$$

Equations (3.6), (3.8) and (3.10) are generated for every couple of exams $[i,j]$ that have students in common. For each couple $[i,j]$, a binary variable C_{ij} is created to determine whether a penalty applies for this couple.

We aim at reducing the number of these variables by creating a variable for each exam instead of each couple. This variable should determine the penalty implied by the exam. This new linearization introduces three sets of variables R_i, D_i and P_i and a parameter T_i.

$$\forall i \in E \quad T_i = \sum_{[i,j] \in A_C} w_{ij}^C \tag{3.12}$$

The parameter T_i represents the number of students exam i shares with other exams. The variables R_i, D_i and P_i represent the penalty implied on placing the adjacent exams of exam i in the periods penalized by the spacing constraints.

$$C^{2R} = w^{2R} \sum_{i \in E} R_i \tag{3.13}$$

$$\left.\begin{array}{c} \forall i \in E \quad \forall p \in P \quad y_{p(p+1)} = 1 \\ R_i \geq (\sum_{[i,j] \in A_C} w_{ij}^C X_{j(p+1)}^P) - T_i(1 - X_{ip}^P) \end{array}\right\} \tag{3.14}$$

Equations (3.14) ensure that, when exam i is placed in the period p (i.e. $X_{ip}^P = 1$), the two-in-a-row penalty is considered for all the adjacent exams in the period $p+1$. $R_i = 0$ when exam i is not placed in period p (i.e. $X_{ip}^P = 0$) because of T_i.

$$C^{2D} = w^{2D} \sum_{i \in E} D_i \tag{3.15}$$

$$\left.\begin{array}{c} \forall i \in E \quad \forall p \in P \\ D_i \geq (\sum_{\substack{q \in P, \\ y_{pq}=1}} \sum_{|p-q|>1} \sum_{[i,j] \in A_C} w_{ij}^C X_{jq}^P) - T_i(1 - X_{ip}^P) \end{array}\right\} \tag{3.16}$$

For Equations (3.16) and (3.16), when exam i is placed in period p, the two-in-a-day and period-spread penalties should be considered for all the periods, i.e. $p+2$ (and $p+3$ for D4 day) for two in a day and all periods within the g^{PS} interval for period spread.

$$C^{PS} = \sum_{i \in E} P_i \tag{3.17}$$

$$\left.\begin{array}{c} \forall i \in E \quad \forall p \in P \\ P_i \geq (\sum_{\substack{q \in P \\ |p-q| \leq g^{PS}}} \sum_{[i,j] \in A_C} w_{ij}^C X_{jq}^P) - T_i(1 - X_{ip}^P) \end{array}\right\} \tag{3.18}$$

$$R_i, D_i, P_i, T_i, C^{2R}, C^{2D}, C^{PS} \in \mathbb{N} \tag{3.19}$$

Note that equations (3.14), (3.16) and (3.18) are generated for every exam i and not for a couple of exams $[i, j]$.

Using this linearization, the total number of variables and constraints in the model is reduced. The number of variables is in the order of n^E and the number of constraints is in the order of $n^E * n^P$.

3.4 A Constraint Programming Approach

We propose a constraint programming model to solve the problem. The aim of this model is to compare the constraint programming and mixed integer programming approaches and check whether it is possible to rapidly obtain good feasible solutions using the solvers.

3.4.1 Representation

The idea is to exploit both the capacities of domains' representation and the available operators that the constraint programming paradigm can offer.

We introduce X_i, an integer variable whose value codes the scheduling of exam i. The integer variable X_i belongs to a restraint domain. The different values of this domain are of the form DDPPRR where:

- DD, the first two digits represent the day
- PP the next two digits represent the period
- RR, the last two digits represent the room where exam i is allocated to at period PP

Note that, the two digits PP correspond to the absolute period number, numbered from 0 to n^P-1, but not to the period number relative to the day DD.

For example, X_i = 020502 signifies that exam i is scheduled on day number 2, period number 5 in room 2. Thus, if the instance disposes 7 days, 36 periods and 7 rooms in each period, the biggest value X_i can take is 073607. Note that the domain of each variable can easily be determined.

In the same spirit, we also introduce the following data:

- DC_d as DD, the code of day d
- PC_p as DDPP, the code of the period p, that belongs to day d
- RC_{pr} as DDPPRR, as the code the room r, of period p, that belongs to day d

Note that they are many codes for a room r according to the day in which it can be used.

Using this representation implies that each period and room have a unique *code*. The room 5 in period 4 in day 3 has the code 030405. Note that the code of r depends on the period in which it is placed. This representation of the assignation of exams is possible with the flexibility of the constraint programming syntax. The period in which exam i is placed can be obtained by a simple integer division by 100. The room is obtained using the remainder of the same division and the day is obtained using an integer division by 10^4. We use the symbol / for integer division, % for the rest of the division and $\equiv$ for equality between two variables or a variable and a parameter. For the constraint programming model, we use the same set of parameters defined in Section 2.3. All the parameters, sets and notations related to rooms, periods and exams remain the same. We invite the reader to refer to Section 2.3 for a detailed descriptions of the parameters, sets and notations. The constraint programming model we used is the following:

3.4.2 Hard Constraints

Room capacities should be respected:

$$\forall p \in P \quad \forall r \in R \quad \sum_{i \in E} s_i^E (X_i \equiv RC_{pr}) \leq s_r^R \tag{3.20}$$

The duration of an exam should be compatible with the duration of the period it is placed in:

$$\forall i \in E \quad \forall p \in P \quad d_i^E (X_i \equiv PC_p) \leq d_p^P \tag{3.21}$$

For every precedence constraint between i and j, the period in which i is placed should be after the period in which j is placed:

$$\forall (i,j) \in H^{aft} \quad (X_i/100) > (X_j/100) \tag{3.22}$$

For every coincidence constraint between i and j, they should be placed in the same period:

$$\forall [i,j] \in H^{coin} \quad (X_i/100) = (X_j/100) \tag{3.23}$$

Two conflicting exams i and j should not be placed in the same period:

$$\forall [i,j] \in B \quad (X_i/100) \neq (X_j/100) \tag{3.24}$$

A room exclusive exam should be alone in the room it is placed in:

$$\forall i \in E^{sole} \quad \forall j \in E \quad \text{such that } i \neq j \qquad X_i \neq X_j \tag{3.25}$$

3.4.3 Soft Constraints

Maximize:

$$C^{2R} + C^{2D} + C^{PS} + C^{FL} + C^{RP} + C^{PP} + C^{NMD} \tag{3.26}$$

Two exams that share students induce a two-in-a-row penalty when placed in the same day back to back:

$$C^{2R} = w^{2R} \sum_{[i,j] \in A_C} w^C_{ij}(X_i/10^4 \equiv X_j/10^4 \ \wedge \ \mid X_i/100 - X_j/100 \mid \equiv 1) \tag{3.27}$$

Two exams that share students induce a two-in-a-day penalty when placed in the same day separated by at least one period:

$$C^{2D} = w^{2D} \sum_{[i,j] \in A_C} w^C_{ij}(X_i/10^4 \equiv X_j/10^4 \ \wedge \ \mid X_i/100 - X_j/100 \mid > 1) \tag{3.28}$$

Two exams that share students induce a period-spread penalty when the gap between the periods in which they are placed is less than g^{PS}:

$$C^{PS} = \sum_{[i,j] \in A_C} w^C_{ij}(\mid X_i/100 - X_j/100 \mid < g^{PS}) \tag{3.29}$$

Front load exams induce a penalty when placed in the front load periods:

$$C^{FL} = w^{FL} \sum_{i \in E^{FL}} (X_i/100 \geq g^{FL}) \tag{3.30}$$

Each exam induce the penalty of the room in which it is placed:

$$C^{RP} = \sum_{r \in R} w^r \sum_{i \in E} (X_i \% 100 \equiv r) \tag{3.31}$$

Each exam induce the penalty of the period in which it is placed:

$$C^{PP} = \sum_{p \in P} w^p \sum_{i \in E} (X_i/100 \equiv PC_p) \tag{3.32}$$

If a room contains multiple exams, at least two of them have different periods, a penalty applies:

$$\forall p \in P \quad \forall r \in R \quad C_{pr}^{NMD} = \sum_{d \in D} (\sum_{i \in E^d} (X_i \equiv RC_{pr}) > 1) \tag{3.33}$$

$$C^{NMD} = \sum_{p \in P} \sum_{r \in R} \quad (C_{pr}^{NMD} - 1) \times (C_{pr}^{NMD} > 0) \tag{3.34}$$

3.5 Results

All tests are performed using CPLEX optimization studio 12.5.1 (IBM [2012]), gcc version 4.5.1, on a machine with an Intel Xeon E5430QC@2.66 GHz CPU and 8 GB of RAM.

Table 3.2 shows the characteristics of the instances of the examination timetabling track in the second International Timetabling Competition (ITC2007). The density d_C for a $G(E, A)$ graph is computed using $d_C = \frac{2|A|}{n(n-1)} \times 100$. n^E represents te number of exams and $\mid A_C \mid$ the number of edges in $G(E, A_C)$. n^S is the number of students, n^P the number of periods and n^R the number of rooms. The PHC (Period Hard Constraint) column corresponds to the sum of the precedence, coincidence and exclusion constraints, while the RHC (Room Hard Constraint) column shows the number of room exclusive constraints.

Tables 3.3 shows the optimal values for the frontload (C_a^{FL}), the period penalty (C_a^P), the room penalty (C_a^R) and the non-mixed-duration penalty (C_a^{NMD}). Columns Opt report the optimal solutions found, Columns UB report the best value found so far in the literature and columns t report the computing times in seconds. We provide model $\mathcal{M}$ with initial solutions obtained using the solver presented in Müller (2009). We set a limit of one day for the computing times. In most cases, the end users can gain useful information for making decisions in the scope of the time spent to build a timetable. Further work has to be done to deal with C^R for instance 1 and C^P for instance 6. In these two cases the time limit has been reached: we reported the CPLEX lower bound and the best integer solution.

It can sometimes be useful for end users to know whether a null or lowest-cost solution can be found for each of these criteria. It can be remarked that all the $C_a^{NMD} = 0$, and one can try to search for a solution without Non-mixed-duration

	d_C	n^E	$\|A_C\|$	n^S	n^P	n^R	PHC	RHC
1	5.05	607	9287	7891	54	7	12	0
2	1.17	870	4421	12743	40	49	12	2
3	2.62	934	11410	16439	36	48	170	15
4	15.0	273	5568	5045	21	1	40	0
5	0.87	1018	4500	9253	42	3	27	0
6	6.16	242	1795	7909	16	8	23	0
7	1.93	1096	11595	14676	80	15	28	0
8	4.55	598	8120	7718	80	8	20	1
9	7.84	169	1113	655	25	3	10	0
10	4.97	214	1133	1577	32	48	58	0
11	2.62	934	11410	16439	26	40	170	15
12	18.45	78	554	1653	12	50	9	7

Table 3.2 – Characteristics of the instances of the examination timetabling track in the second International Timetabling Competition (ITC2007).

	C_a^{FL}			C_a^P			C_a^R			C_a^{NMD}		
	Opt_a	UB	t	Opt_a	UB	t	Opt_a	UB	t	Opt_a	UB	t
1	125	255	3042	0	270	1002	0/350	1050	-	0	100	200
2	0	375	8255	0	0	249	0	0	312	0	0	277
3	30	740	2218	0	100	384	0	0	305	0	0	126
4	25	105	6341	50	1750	20648	0	0	60	0	0	72
5	50	1440	657	0	100	42	0	0	13	0	0	45
6	375	375	25	30/55	450	-	950	1200	331	0	75	1519
7	0	460	250	0	0	339	0	0	232	0	0	37
8	0	380	259	0	342	196 8	0	125	463	0	0	41
9	0	0	4	0	0	1	0	0	1	0	0	1
10	0	5	7	0	0	4	0	0	2	0	0	6
11	30	1370	7514	0	0	509	0	0	224	0	0	145
12	0	40	1	0	0	1	0	0	1	0	0	1

Table 3.3 – Optimal and best values from the literature for Front Load, Period Penalty, Room Penalty and Non-Mixed Duration

	$(C_b^{2R} + C_b^{2D})$			C^{PS}				
	LB	UB	t	LB	UB	t	LB	Best
1	0	42	5	126	2534	4130	215	4128
2	10	10	79	0	0	0	10	380
3	330	2885	82095	140	4926	4645	501	7769
4	291	9104	52632	47	3925	129	428	13103
5	0	0	nc	142	1259	742	136	2513
6	1740	3700	17223	19900	19900	84	22995	25330
7	0	0	nc	312	3602	3449	220	3537
8	0	0	0	836	6718	4837	552	7087
9	0	0	2	57	576	5	57	913
10	0	300	1	2786	11585	4	2786	13053
11	2660	10690	22566	620	13079	5324	3310	24369
12	1330	3780	1	541	1901	3	1871	5095

Table 3.4 – Lower bounds for $(C_b^{2R} + C_b^{2D})$ and C^{PS} and the gap between the lower bounds and the best solutions in the literature.

violations. The optimal $C_a^P = 0$ allows end users to remove the penalized periods prior to the solution process. For Instance 4, there are two large period penalties (200 and 500), and the corresponding periods can also be removed, since a $C_a^P = 50$ solution is attainable. In the initial dataset, Room penalties are imposed for instances 1, 2, 3, 6, and 7. The optimal solutions $C_a^R = 0$ may be used by end users to avoid all the penalized rooms. The optimal solution can be seen to be close to the best value for Instance 6, and here a large room penalty cannot be avoided. Considering the C_a^{FL} criterion, the value is tightened for Instance 6 ($Opt_a = UB$).

Table 3.4 displays the values of lower bounds we obtained and the best values from the literature for $(C_b^{2R} + C_b^{2D})$ and C^{PS}. We use Östergård (2002) to compute all the cliques larger than the considered limits. For Instances 2, 5, 7 and 8 the bounds are tightened for $(C_b^{2R} + C_b^{2D})$. Columns t report the global computing time: clique computing, evaluation of each clique using the model $\mathcal{K}$ and the greedy algorithm. Note that for instances 5 and 7, we have $L^{2RD} < \omega_{A_{GC}}$: there is no clique that can be used to compute, the value is zero and we report nc in column t. The evaluation of a clique using the model $\mathcal{K}$ is very fast, but one can have a large number of cliques (e.g. the number of cliques is 15598206 for Instance 3). For Instance 6 the best value for C^{PS} is achieved using the lower bound

computation: this is the optimal value for this particular instance where $g^{PS} > n^P$ (see Gogos, Alefragis, and Housos (2012)). The computing time depends on the considered instance for the two lower bounds since they are tightly coupled to the number of cliques larger than the limit.

Penultimate Column *LB* (see Table 3.4) reports the sum of the contributions of the optimal values and of the obtained lower bounds, while column *Best* reports the values of the best solutions found so far (Hamilton-Bryce, McMullan, and McCollum [2014]). As it can be noticed, these problems remain challenging.

	MIP Model			CP Model
Instance	Root Node Time	LB	UB	UB
Instance 1	-	0	NS	22568
Instance 2	1810	40.66	NS	50569
Instance 3	-	0	NS	108985
Instance 4	65	25.35	NS	25717
Instance 5	46	348.96	361960	72140
Instance 6	968	1627.17	NS	34570
Instance 7	-	0	NS	50115
Instance 8	-	0	NS	53752
Instance 9	5	5.45	2790	2181
Instance 10	10	0	17941	62257
Instance 11	-	0	NS	173235
Instance 12	1	30	5826	12429

Table 3.5 – Results of the MIP and CP models within one-hour time limit

To compare the effect of the MIP model with the new linearization against the Constraint Programming (CP) model, Table 3.5 presents the results of both models. The models were run in a one-hour time limit. Columns UB show the best solution found by the corresponding model. Column "Root Node Time" shows the time needed to solve the LP relaxation for the new MIP model. When the LP relaxation is not solved within the time limit, the instance is flagged by "-". Column "LB" shows the lower bound by the MIP model within the time limit.

The table shows that the CP model is able to find feasible solutions for all the instances. The MIP model faces considerable difficulties solving the LP relaxation. It finds a feasible solution for four instances (5, 9, 10 and 12). One can easily notice that some instances are quite difficult to solve. The CP model is clearly out-

performing the MIP model in finding feasible solutions. It is unfortunately not possible to compare the lower bound results on both models since the CP solver proves optimality by proving that no better solutions exist.

The quality of the feasible solutions found by the CP model is far from the quality of the best solutions found in the literature. This is justified by the fact that the constraint programming is a generic approach whereas the best solutions were found using dedicated heuristic and metaheuristic approaches. The use of mathematical and constraint programming in exam timetabling problems is limited due to the big number of variables and constraints. The solvers are usually unable to provide better solutions than the ones provided by problem-adapted approaches. The use of available solvers to solve mixed integer programming and constraint programming models has the disadvantage of the inability to adapt the algorithm to the problem specifications.

3.6 Conclusion

We presented a set of approaches to compute lower bounds, to reduce the number of variables and constraints for the mixed integer model and to provide a constraint programming model for the exam timetabling problem. The lower bounds are computed for the Two In a Row, Two In a Day and Period Spread soft constraints based on the cliques that can be extracted from the conflict graph. To the best of our knowledge, this is the first lower-bound-related work for the ITC2007 exam timetabling problem. We used the improved model presented in the previous chapter to obtain optimal solutions when considering the the Front Load, Non Mixed Duration, Period and Room penalties individually. We also provided a new formulations for the Two In a Row, Two In a Day and Period Spread soft constraints that make it possible to reduce the number of variables constraints for the MIP model. The Constraint Programming model presented was able to find feasible solutions for all the instances of the benchmark and can be hybridized with the MIP model to provide starting solutions. The work presented provides an promising beginning in exploiting mathematical and constraint programming as solving techniques for the exam timetabling problem.

CHAPTER 4

A MEMETIC APPROACH FOR EXAM TIMETABLING

Contents

4.1 Introduction

The exam timetabling problem represents one of the main problems encountered every year in academic institutions. The problem consists in assigning a set of exams to a set of periods and rooms while respecting hard constraints. A solution is feasible if all the hard constraints are met. To evaluate the quality of the timetable, soft constraints can be introduced. When a soft constraint is violated, a corresponding penalty is applied. Best solutions are therefore the ones that minimize the soft constraint penalties, i.e. the overall cost.

Researchers have shown practical interest in solving exam timetabling problems. However, comparing the different methods they proposed is difficult since institutional constraints change from one university to another. Academic benchmarks were proposed in the literature to provide the community with a common test base on which methods can be tested. The Toronto benchmark (Carter, Laporte, and Lee [1996]), Nottingham benchmark (Burke, Newall, and Weare [1996]), Melbourne benchmark (Merlot et al. [2003]) and the second international competition (*Metaheuristics Network.* [2002]; McCollum et al. [2010a]) are the most used benchmarks so far. The Toronto benchmark is a collection of thirteen instances taken from real-world universities across the world. The Nottingham and Melbourne benchmarks are a set of instances taken from the Nottingham and Melbourne universities. The second international timetabling competition provides a set of twelve instances taken from different universities.

The core of many exam timetabling problems is a graph coloring problem. However, the constraints to be satisfied differ from one benchmark to another. For instance, room-related constraints are present only in one of the variants of the Toronto benchmarks. The second International Timetabling Competition (ITC2007) introduces exam-related hard constraints (e.g. precedence and coincidence between exams) which are not included in other benchmarks. Considering the real-world constraints it introduces, the ITC2007 benchmark is considered as a solid test base on which methods can be tested.

Evolutionary algorithms are some of the most used and successful approaches applied to exam timetabling. Burke, Newall, and Weare (1996) used a memetic algorithm to solve the Toronto and the Nottingham benchmarks. The authors used two stages: local search and mutation. The local search stage used the Hill-

Climbing heuristic. The mutation stage is done using two operators: *Heavy and Light Mutation operators*.

Abdullah et al. (2010) used a tabu-based memetic scheme. The population initialization stage used a saturation degree graph coloring heuristic. Once the crossover and the mutation stages are done, an improvement stage is used to improve the quality of the child. For this purpose, a set of neighborhood structure are used. Tabu search was used in the improvement stage to penalize the neighborhood structures that failed to improve the child. The authors claimed that using multiple neighborhood structures proved to be effective in better exploring the search space.

Ülker, Özcan, and Korkmaz (2007) investigated Linear Linkage Encoding for Grouping problems and tested it on graph coloring and exam timetabling. The author modeled the exam timetabling problem as a grouping problem. The exams are the items and the periods represent the groups in which items are grouped. The algorithm used two crossovers, the Lowest Index First Crossover and the Lowest Index Max Crossover to create a new chromosome. We refer to Qu et al. (2009a) for a detailed overview on the techniques used in exam timetabling.

We propose in this chapter a memetic approach for the exam timetabling problem encountered at the Université de Technologie de Compiègne (UTC). The exam timetabling problem at UTC includes some of the widely encountered hard and soft constraints and some problem-specific constraints. These constraints however fall in the potential extensions of the ITC2007 exam timetabling track described by McCollum et al. (2012).

The remainder of the paper is as follows. Section 4.2 presents the exam timetabling problem at UTC. We show how the UTC's problem represents a further extension of the ITC2007 problem. We believe the instances taken from UTC can be a consolidating set to be added to the ITC2007 instances to provide the community with a real-world exam timetabling benchmark. Section 4.3 introduces the mathematical model used formulate the problem and validate the solutions obtained by the approach. Section 4.4 presents the memetic approach and details its components. The results are analyzed and discussed in Section 4.5. Finally, the conclusions and future work are to be found in Section 4.6.

4.2 Problem description

The exam timetabling problem at the UTC has evolved since the university was created. In the sequel, we present the problem encountered nowadays. The soft and hard constraints are named using the terminology used in the ITC 2007 in order to have a unified terminology.

When we first met with the practitioners, they explained the exam timetabling at UTC like a classical exam timetabling problem. After several meetings, we found out that a number of constraints are not classical and have not been considered in the benchmarks. For instance, UTC has two different sites separated by several kilometers. This situation will change in near future because new buildings are being built. Thus, the practitioner has to deal with the travel of students between sites if two exams for the same student are scheduled back to back. The practitioners also informed us that splitting exams into different rooms is allowed at UTC.

That being said, the exam timetabling problem at UTC consists in assigning exams to a set of periods and rooms while respecting the hard constraints. Exams are divided into two sets: splittable and non-splittable exams. Each exam has a set of allowed periods and a set of allowed rooms. Periods can overlap and rooms are not available at all periods. Thus, a list of available rooms is associated to each period.

The hard constraints are the following:

- A student cannot sit two exams at the same period or at two overlapping periods.
- An exam must be assigned to a unique period.
- A non-splittable exam must be assigned to a unique room.
- The duration of the exam must be less than or equal to the duration of the period in which it is assigned.
- The capacity of any room should not be exceeded at any period.
- The sum of the numbers of students in portions of a splittable exam should be equal to the total number of enrolled students.

- A room can be used only at one period of a set of overlapping periods.
- Each exam should be assigned only once.

The practitioner considers the timetable feasible when all the hard constraints are met. The quality of the solution is measured using soft constraints. When a soft constraint is not satisfied, a penalty is applied. The soft constraints used to measure the quality of the solution differ from one institution to another. Due to the limited time given to professors after the exams to mark them, the most important soft constraint is to place the exams with a large number of students early in the timetable.

Since exams may take place over different sites, a travel penalty is applied if the student has two exams in a row in different sites. Note that travel penalty is different from the two in a row penalty: a travel penalty does not occur while two in a row does if the student has two exams in a row in the same site.

The following definitions describe briefly the soft constraints used by the practitioner:

Two In a Row: Exams of a student allocated back to back in the same day should be avoided.

Two In a Day: Exams of student scheduled in the same day but not back to back should be avoided.

Front Load: Large-size exams should be assigned before a certain period.

Period Spread: Exams of a student allocated in a certain period gap should be avoided.

Non Mixed Duration: Exams with different durations should not be assigned in the same room at the same period.

Period Penalty: Each period with a cost should be avoided.

Room Penalty: Each room with a cost should be avoided.

Travel Penalty: Exams in row for the same student in different sites should be avoided.

Note that ITC2007 differs in both the hard and the soft constraints. McCollum et al. (2012) gave potential extentions of the ITC2007 problem. Some of the extensions fit in the UTC's problem. For instance, the problem in hands allows splitting exams between rooms. It also considers multiple sites in which exams take place. This case is handled using a soft and a hard constraint: the hard constraint ensures that the students have enough time to travel from one site to another. The soft constraint implies a penalty (travel penalty) when a student has two exams in a row in the same day in different sites. Room and period preferences are also introduced in UTC's problem. An exam has a list of "allowed" periods and rooms in which it can be placed. At UTC, the room availability across periods is an important factor in the exam timetabling problem. Due to management reasons, some rooms are not available for more than a week. Therefore, the set of available rooms changes from one period to another.

Instance	n^E	n^S	n^P	n^R	Conflict density
S2011	141	2322	32	8	0.30
F2011	119	2388	24	9	0.32
S2012	142	2412	36	9	0.31
F2012	117	2296	26	8	0.30

Table 4.1 – Characteristics of UTC instances

Table 4.1 presents the characteristics of four instances relative to four semesters in UTC. Column *instance* reports the labels where "S" stands for the spring semester and "F" stands for the fall semester. Columns n^E, n^S, n^P, n^R and *Conflict density* show for each instance the number of exams, the number of students, the number of periods, the number of rooms and the density of conflict graph $G(E, B)$, respectively.

4.3 Mathematical model

We introduce the mathematical model used to formulate the problem and to assess the set of solutions obtained by the memetic approach.

4.3.1 Parameters

Exam Parameters

E: Set of exams, $n^E = |E|$ is the number of exams.

E^S: Set of splittable exams.

E^{NS}: Set of non-splittable exams.

E^{Exc}: Set of exams subject to *Room Exclusive* constraints.

B: Set of edges $[i, j]$ where i and j are two exams who have students in common.

s_i^E: Number of students for the exam $i \in E$.

d_i^E: Duration of the exam $i \in E$.

E^d: Set of the exam $i \in E$ with duration d.

D: Set of durations.

w_{ij}: Number of students in common between exams i and j.

E^{FL}: Set of exams subject to the Front Load penalty and n^{FL} is the number of exams in this set.

P_i^E: Set of periods allowed for exam i.

$G(E, B)$: Conflict graph.

$\mathcal{N}(i)$: Set of the neighbors of exam i in the conflict graph G.

Room Parameters

S: Set of sites in the university and $n^S = |S|$ is the number of sites in this set.

R: Set of all rooms available and $n^R = |R|$ is the number of rooms in this set.

R_s: Set of rooms of site s.

P_r^R: Set of periods allowed for room r.

s_r^R: Capacity of room $r \in R$.

n_p^R: Number of available rooms in period p.

w_r^R: Penalty weight of using room r.

Period Parameters

P: Set of periods and n^P is the number of periods in this set.

P_{ir}: Set of allowed periods for exam i and room r, $P_{ir} = (P_i^E \cap P_r^R)$.

d_p^P: Duration of period $p \in P$.

I_p: Time interval of period $p \in P$. Time intervals are used for overlapping periods. If two periods p and q overlaps, then $I_p \cap I_q \neq \emptyset$

$\gamma(p)$: Set of periods immediately after period p in the same day.

$\Gamma(p)$: Set of periods not immediately after period p in the same day.

w_p^P: Penalty weight of using period p.

4.3.2 Variables

Primary Decision Variables

X_{ipr}: a boolean, 1 *iff* exam i or a portion of it is placed at period p in room r.

X_{ip}: a boolean, 1 *iff* exam i is placed at period p.

Secondary Decision Variables

A_{ipr}: An integer representing the number of students of a splittable exam $i \in E$ placed at period p in room r.

C_{ij}^{2R}: A boolean, set to 1 *iff* exams i and j, having students in common, are placed at two periods back to back in the same day.

C_{ij}^{2D}: A boolean, set to 1 *iff* exams i and j, having students in common, are placed at two periods not back to back in the same day.

C_{ij}^{PS}: A boolean, set to 1 *iff* exams i and j, having students in common, are placed at two periods spread by a gap less than the defined *gap of period spread*.

C_{pr}^{NMD}: An integer representing the number of different periods assigned in room r at period p.

U_{dpr}^{D}: A boolean, set to 1 if one or multiples exams of duration d are assigned at period p in room r.

C_{ij}^{T}: A boolean, set to 1 *iff* exams i and j have student in common and are placed in two periods back to back and in rooms located at different sites.

Institutional Parameters

w^{2R}: Weight of the Two In a Row penalty.

w^{2D}: Weight of the Two In a Day penalty.

w^{T}: Weight of the Travel penalty.

w^{NMD}: Weight of the Non Mixed Duration penalty.

g^{PS}: Gap of the Period Spread penalty.

w^{FL}: Weight of the Frontload penalty.

T^{FL}: Threshold of the number of students for an exam to be considered for the Front Load penalty. We have: $\forall i \in E^{FL} \quad s_i \geq T^{FL}$.

g^{FL}: The period number starting from which the Front Load penalty is considered. If, for a period p, $g^{FL} \prec p$, then p is considered as a Front Load period.

Even though the notations are close to those of ITC2007, some differences can be noted. Due to overlapping periods, g^{FL} is the number of a particular period instead of the total number periods. The set of large exams E^{FL} and therefore the number of large exams can be tuned by the user by changing the value of T^{FL}. For instance, by setting $T^{FL} = 100$, we get in E^{FL} the exams having a number of students greater than or equal to 100.

4.3.3 Formulation

The objective function

Minimize :

$$C^{2R} + C^{2D} + C^{PS} + C^{FL} + C^{NMD} + C^{P} + C^{R} + C^{T} \tag{4.1}$$

The objective function is a sum of penalty terms. Each of the terms refers to a specific soft constraints.

Hard Constraints

The following hard constraints condition the feasibility of the solution.

The sum of splittable exams (or portions) allocated to room r at period p should not exceed the capacity of the room:

$$\forall r \in R \quad \forall p \in P_r^R \quad \sum_{i \in E^S \ / \ p \in P_i^E} A_{ipr} \leq s_r^R \tag{4.2}$$

Linking the variables X_{ipr} and A_{ipr} relative to splittable exams:

$$\forall i \in E^S \quad \forall r \in R \quad \forall p \in P_{ir} \quad \begin{cases} A_{ipr} \leq s_i^E X_{ipr} \\ A_{ipr} \geq X_{ipr} \end{cases} \tag{4.3}$$

The two parts of Equations 4.3 are required to check $X_{ipr} = 1$ *iff* $A_{ipr} \neq 0$.

The sum of the portions of a splittable exam should be equal to the number of enrolled students:

$$\forall i \in E^S \quad \sum_{r \in R} \quad \sum_{p \in P_{ir}} A_{ipr} = s_i^E \tag{4.4}$$

Linking the variables X_{ipr} and X_{ip} relative to splittable exams:

$$\begin{array}{l} \forall i \in E^S \\ \forall p \in P_i^E \end{array} \quad \begin{cases} \displaystyle\sum_{r \in R \ / \ p \in P_{ir}} X_{ipr} \leq n_p^R X_{ip} \\ \displaystyle\sum_{r \in R \ / \ p \in P_{ir}} X_{ipr} \geq X_{ip} \end{cases} \tag{4.5}$$

At most n_p^R rooms can be used to schedule exams in period p. Equations (4.5)

ensures that, when $X_{ip} = 1$, there is at least one variable X_{ipr} set to to one and at most n_p^R. Every exam should be placed in exactly one period:

$$\forall i \in E \quad \sum_{p \in P_i^E} X_{ip} = 1 \tag{4.6}$$

A non splittable exam should be assigned in exactly one room:

$$\forall i \in E^{NS} \quad \sum_{p \in P_i^E} \sum_{r \in R} X_{ipr} = 1 \tag{4.7}$$

The sum of the sizes of non splittable exams allocated to room r at period p should not exceed the size of room r:

$$\forall r \in R \quad \forall p \in P \quad \sum_{i \in E^{NS}} s_i^E X_{ipr} \leq s_r^R \tag{4.8}$$

A room cannot be used by two exams in two overlapping periods:

$$\left.\begin{array}{l} \forall i, j \in E \quad \forall r \in R \\ \forall p, q \in (P_{ir} \cap P_{jr}) \\ I_p \cap I_q \neq \emptyset \quad p \neq q \end{array}\right\} \quad X_{ipr} + X_{jqr} \leq 1 \tag{4.9}$$

This hard constraints is added due to considering multiple sites that cannot be found in the ITC2007 problem.

Two exams that have students in common cannot be placed in the same period or in two overlapping periods:

$$\forall [i, j] \in B \quad \forall p \in P_i^E \quad \forall q \in P_j^E \quad I_p \cap I_q \neq \emptyset \qquad X_{ip} + X_{jq} \leq 1 \tag{4.10}$$

Every exam subject to the room exclusive constraint should be on its own:

$$\forall i \in E^{Exc} \quad \forall j \in E \quad i \neq j \quad \forall r \in R \quad \forall p \in (P_{ir} \cap P_{jr}) \quad X_{ipr} + X_{jpr} \leq 1 \tag{4.11}$$

When the exam subject to the room exclusive constraint is splittable, each portion should be on its own.

Soft Constraints

The quality of the solution is measured using the following soft constraints.

Whenever two exams who have students in common are placed in two consecutive periods in the same day, a two in a row penalty applies:

$$C^{2R} = w^{2R} \sum_{[i,j] \in B} w_{ij} C_{ij}^{2R} \tag{4.12}$$

$$\left.\begin{array}{l} \forall [i,j] \in B \quad \forall p \in P_i^E \\ \forall q \in P_j^E \cap \gamma(p) \end{array}\right\} X_{ip} + X_{jq} \leq 1 + C_{ij}^{2R} \tag{4.13}$$

Whenever two exams who have students in common are placed in two non-consecutive periods in the same day, a two in a day penalty applies:

$$C^{2D} = w^{2D} \sum_{[i,j] \in B} w_{ij} C_{ij}^{2D} \tag{4.14}$$

$$\left.\begin{array}{l} \forall [i,j] \in B \quad \forall p \in P_i^E \\ \forall q \in P_j^E \cap \Gamma(p) \end{array}\right\} X_{ip} + X_{jq} \leq 1 + C_{ij}^{2D} \tag{4.15}$$

Whenever two exams who have students in common are placed in two periods between which the gap is less than the period spread gap, a period spread penalty applies:

$$C^{PS} = \sum_{[i,j] \in B} w_{ij} C_{ij}^{PS} \tag{4.16}$$

$$\left.\begin{array}{l} \forall [i,j] \in B \quad \forall p \in P_i^E \\ \forall q \in P_j^E \text{ and } p \prec q \\ \text{such that } \mid p - q \mid \leq g^{PS} \end{array}\right\} X_{ip} + X_{jq} \leq 1 + C_{ij}^{PS} \tag{4.17}$$

If a front load exam is placed in a front load period, a front load penalty applies:

$$C^{FL} = \sum_{i \in E^{FL}} \quad \sum_{p \in \{t / t \in P_i^E,\, g^{FL} \prec t\}} X_{ip} \tag{4.18}$$

If a room with a non-null cost is used, the room penalty applies:

$$C^R = \sum_{i \in E} \sum_{p \in P} \sum_{r \in R} w_r^R X_{ipr} \tag{4.19}$$

If a period with a non-null cost is used, the period penalty applies:

$$C^P = \sum_{i \in E} \sum_{p \in P} \sum_{r \in R} w_p^P X_{ipr} \tag{4.20}$$

Due to multiple sites in which exams take place, whenever two exams that have students in common are placed in two consecutive periods in the same day and in two different sites, the travel penalty applies:

$$C^T = w^T \sum_{[i,j] \in B} w_{ij} C_{ij}^T \tag{4.21}$$

$$\left.\begin{array}{l} \forall [i,j] \in B \quad \forall p \in P_i^E \\ \forall q \in P_j^E \cap \gamma(p) \quad \forall s, s' \in S \end{array}\right\} \sum_{r \in R_s} X_{ipr} + \sum_{r' \in R_{s'}} X_{jqr'} \leq 1 + C_{ij}^T \tag{4.22}$$

Note that Equations (4.13) differ from equations (4.22) since we consider rooms located at a particular site in Equations (4.22).

Whenever exams with different durations are placed in the same room at the same period, a non mixed duration penalty applies:

$$\left.\begin{array}{c} \forall d \in D \quad \forall p \in P \quad \forall r \in R \\ |E^d| U_{dpr}^D \geq \sum_{i \in E^d} X_{ipr}^{PR} \end{array}\right\} \tag{4.23}$$

$$\forall p \in P \quad \forall r \in R \quad 1 + C_{pr}^{NMD} \geq \sum_{d \in D} U_{dpr}^D \tag{4.24}$$

$$C_{pr}^{NMD} \geq 0 \tag{4.25}$$

$$C^{NMD} = w^{NMD} \sum_{p \in P} \sum_{r \in R} C_{pr}^{NMD} \tag{4.26}$$

$$\begin{aligned} X_{ip}, X_{ipr}, C_{ij}^{2R}, C_{ij}^{2D}, C_{ij}^{PS}, C_{ij}^{T}, U_{dpr}^{D} &\in \{0,1\} \\ C^{2R}, C^{2D}, C^{PS}, C^{FL}, C^{NMD}, C^{P}, C^{R}, C^{T}, A_{ipr} &\in \mathbb{N} \end{aligned} \tag{4.27}$$

The model of equations (4.1) to (2.26) is useful to describe our problem. It permits to formally model the novel hard and soft constraints. This MIP was coded to check solutions built using the memetic approach presented in the next section. Unfortunately, it cannot be used to obtain good solutions in a reasonable time. A week of computation cannot permit to obtain solutions better than those obtained by the memetic approach within an order of magnitude less time.

4.4 Memetic algorithm

As claimed by Qu et al. (2009b), evolutionary algorithms give some of the best results for different benchmarks. Memetic algorithms (MA) come from a combination of Genetic Algorithms and Local Search methods. We used a memetic algorithm to solve our problem. The reason for this choice is the diversity and the efficiency of population-based techniques. The key feature of MA is in coupling the global optimization done by the the crossover procedure and the local optimization that enhances the solutions explored. A population of individuals that evolves throughout a series of crossovers is used. Each individual of the population is called a *chromosome*. A chromosome can represent one or multiple solutions depending on the encoding used. A direct encoding signifies that the solution is directly represented by the chromosome and that no decoding procedure is needed. An indirect encoding however means that a decoding procedure is needed to retrieve the solution from the chromosome. Our algorithm is based on an *indirect encoding*. One of the advantages of using an indirect encoding is allowing one chromosome to represent multiple solutions, i.e. a neighborhood of solutions. In the sequel, we present the implementation details of our MA.

4.4.1 Solution Representation and Decoding/Encoding procedure

Each solution for the problem is represented by a chromosome that belongs to the population. A chromosome is a permutation of integers, each of which represents the number of an exam that has to be assigned in to least one room and to one

period.

Given a permutation of integers, we apply the *first fit decoding (FFD)* to obtain a solution. The FFD is inspired by the the Bin Packing heuristic First fit algorithm. Exams in the permutation are taken in turn and assigned to the first available room and period that respects the hard constraints. FFD proved to be efficient and fast. During the decoding, the solution cost is assessed. If the chromosome leads to an infeasible solution because some exams are left unscheduled, a destruction/construction *repairing method* is applied to obtain a feasible solution and the chromosome is consequently updated.

The periods and rooms are ordered according to their first apparition in the instance file. A permutation (i.e. chromosome) can be obtained back from a solution by taking the exams from this solution in the order of the periods and rooms, i.e. the first exam in the permutation is one of the exams in the first period at the first room. When all exams in the first room are considered, the exams in the second room take the role. When all exams in the first period are considered, the same procedure is applied on the remaining periods in the same order until all the permutation is built.

4.4.2 Repairing method

At the end of the decoding process that failed to assign all the exams, the lists of scheduled and unscheduled exams are L_{SCH} and L_{NSCH} respectively. A number k of exams are randomly selected from L_{SCH} and then removed to added to L_{NSCH}. The updated list L_{NSCH} is then shuffled. FFD is applied then applied on L_{NSCH}. If all the exams in L_{NSCH} are scheduled, the solution is reported as valid. If not, the same procedure is repeatedly applied until a valid solution can be obtained. The Repairing Method (RM) is given in Algorithm 4.

Algorithm 3: Repairing Method

Data: L_{SCH}, L_{NSCH}
Result: Valid Solution

1 Select randomly k exams from L_{SCH} and remove them ;
2 Add the removed exams to L_{NSHC} ;
3 Shuffle L_{NSCH} ;
4 FFD(L_{NSCH}) ;

4.4.3 Population and algorithm initialization

The population consists of *pop_size* chromosomes randomly generated are the beginning of the algorithm. If any of chromosomes appears to be invalid during the decoding, RM is invoked. Next, a number *POP_OPT* of chromosomes are improved using a local search. The local search is applied to ensure good quality chromosomes in the population for crossing.

4.4.4 Crossover and population update

At each iteration, a couple of parents are chosen from the population using the Binary Tournament strategy. This strategy randomly selects two couples of chromosomes from the population and then, out of each couple, the chromosome with the least cost is selected. Two parents are therefore selected for the crossover. Next, the Linear Order Crossover (LOX) is applied on the two selected parents to produce a child chromosome. LOX selects a parent and randomly defines two indexes in the permutation of the selected parent. The exams' numbers between the two indexes are then given to the child and the rest of the child is completed from the remaining parent. The child is validated and evaluated. If it is not valid, RM is applied. Since the population should keep a constant size, if the child has the same cost as any chromosome of the population, the existing chromosome is replaced by the child. Otherwise, if the child has better cost than the chromosome with the worst cost of the population, the child replaces it.

4.4.5 Local search operators

Once a child is obtained from the crossover, the local search stage is applied with a probability *p_m*. We use three local search operators: Light Destruction/Construction (LDC), Hill Climbing (HC) and Swap. To decide which operator is applied, a list of Boolean flags is used. A *false* flag signifies that the operator has not been used yet. The stage starts by randomly selecting an operator and applying it on the current solution. If the operator does not improve the chromosome, another operator is randomly selected from the remaining unused operators. If one of the operators improves the chromosome, all the flags (including the flag of the current operator) are set to false. We choose to reset the flags

to give all the operators another chance to improve the new solution. The local search stage ends when the flags of all operators are set to true.

Best Insertion Procedure (BI)

The local search operators defined in the sequel inserts exams in the solution using the Best Insertion procedure. This procedure places the exam in the period and the room that minimize the penalty implied by this exam.

When the exam is not splittable, placing the exam requires a single room and a single period. Thus, for each room in each period, we compute the penalty implied by violating all the soft constraints and we select the room and the period with the minimum penalty.

When the exam is splittable, assigning the exam requires selecting a period and one or multiple rooms. Since the penalties implied by placing the exam on rooms and periods are independent, we start by first selecting the period for which the exam implies the minimum penalty. Two In a Row, Two In a Day, Period Spread, Front Load and Period penalties are the considered penalties when evaluating the penalty related to the period. Next, we seek to find the set of rooms for which the exam implies the minimum penalty for the Non Mixed Duration, Travel and Room penalties. To do so, we model the problem as a knapsack problem. Each room corresponds to an item and the exam represents the knapsack. Each room has a cost that corresponds to the penalty of planning the exam or a portion of it in this room. In the following, we present the formulation of the problem:

Minimize:

$$\sum_{r \in R} L_r^R X_r \tag{4.28}$$

Subject to:

$$\sum_{r \in R} s_r^R X_r \geq s_i^E \tag{4.29}$$

The decision variable $X_r = 1$ when the exam i or a portion of it is placed in the room r. The objective is to minimize the penalty implied by placing exam i in the different rooms. to which it can be assigned. L_r^R represent the penalty implied by placing the exam i in the room r for the three soft constraints (Non

Mixed Duration, Travel and Room soft constraints). To solve the problem, we applied the classical dynamic programming approach. Therefore, the Best Insertion procedure is of complexity $\mathcal{O}(n)$.

Algorithm 4: LDC scheme

```
1  Backup(current_solution) ;
2  Best = current_solution ;
3  while iter < ITER_MAX_HC do
4  |   Remove randomly an exam e and k of its neighbors ;
5  |   Put the removed exams in list of unscheduled exams L_NSCH ;
6  |   Shuffle the list L_NSCH ;
7  |   for each exam e' in L_NSCH do
8  |   |   Place e' using BI ;
9  |   end
10 |   if all exams are placed AND Best > current_solution then
11 |   |   Best = current_solution ;
12 |   end
13 |   ++iter ;
14 end
15 if Best not improved then
16 |   restore the original solution
17 end
```

Light Destruction/Construction (LDC)

This operator consists in removing an exam and k of its adjacent exams in the conflict graph and then trying to better place them. Algorithm 4 presents the algorithm of this operator. The $k+1$ exams are removed and considered as unscheduled. They are shuffled and inserted again in the timetable using the Best Insertion procedure. If some exams cannot be placed, another exam is randomly chosen and k exams of its adjacent exams are removed. The unscheduled exams are shuffled and placed using the Best Insertion procedure. This Destruction/Construction routine is repeated until a number of maximum iterations without improving the solution is reached. LDC is run in $\mathcal{O}(n^2)$-time.

Algorithm 5: Hill Climbing scheme

```
1  Backup (current_solution) ;
2  Best = current_solution ;
3  while iter < ITER_MAX_HC do
4  |  Generate a random permutation of exams ;
5  |  for Each exam e in the permutation do
6  |  |  Remove exam e ;
7  |  |  Insert exam e back using BI
8  |  end
9  |  if Best > current_solution then
10 |  |  Best = current_solution;
11 |  end
12 |  iter++;
13 end
14 if Best not improved then
15 |  restore the original solution
16 end
```

Hill-Climbing (HC)

Hill-Climbing is often used as the core local search of memetic algorithms (Burke, Newall, and Weare [1996]). Algorithm 5 presents the algorithm of HC. The stopping condition of Hill-Climbing is a maximum number of iterations without improvements. At the beginning of each iteration, a random order of the exam is built. Next, Hill-Climbing considers ordered exams in turn and removes them. After the removal of each exam, the Best Insertion is called to reinsert the removed exam. Since the Best Insertion procedure is used to place each exam, HC ensures that, at the end of operator, the solution is valid and at least with the same cost. If HC finds a new solution and its quality is better than the previous solution, the new solution is considered as the best. Otherwise, the previous best solution is considered for the next iteration. As a result, HC is of complexity $\mathcal{O}(n^2)$.

Swap

The Swap operator is widely used on different problems. It is often used to test whether symmetry exists. Swap may be applied to swap periods, rooms or exams. We opted for period swapping. Algorithm 6 presents the algorithm of the

Algorithm 6: Swap scheme

```
1  Backup the current solution ;
2  Best = current_solution ;
3  while iter < ITER_MAX_SWAP do
4  |   Select randomly two periods p and p;
5  |   Swap the exams between the two periods ;
6  |   if Best > current_solution AND solution is valid then
7  |   |   Best = current_solution;
8  |   end
9  |   ++iter ;
10 end
11 if Best not improved then
12 |   restore the original solution
13 end
```

Swap operator. Two periods are randomly selected and the exams scheduled in these periods are swapped. The feasibility and the cost of the solution are then assessed. When the new solution is infeasible or its cost is higher than the older one, it is rejected. If the new solution's cost is better than the best solution, the best solution is updated. The period swapping is of complexity $\mathcal{O}(n^2)$.

4.4.6 Algorithm scheme

The memetic algorithm, presented in Algorithm 7, starts by initializing the population. Chromosomes are generated randomly and *POP_OPT* chromosomes are improved using the local search operators. The population is then sorted with respect to the cost of chromosomes. The LOX crossover is used to generate a new child chromosome at each iteration by crossing two parents from the existing population. It is next improved with a probability *p_m* using local search operators and inserted in the current population if its cost is at least better than the worst chromosome. The approach stops after a maximum number of iterations without improvements. Finally, the head of the population is reported as the best solution.

Algorithm 7: The proposed memetic approach

```
Data: ITER_MAX, p_m
Result: S_best: best solution found
Pop initialization;
iter = 0;
while (iter ≤ ITER_MAX) do
    Choose two parents p_1 and p_2 using Binary Tournament;
    ρ = LOX (p_1,p_2);
    if (ρ is valid) then
        RM(ρ);
    end
    if (rand(0,1) ≤ p_m) then
        apply local search on ρ ;
    end
    if (Evaluation(ρ) ≤ Worst solution) then
        if (∃ Chromosome λ ∈ Pop AND Eval(λ) = Eval(ρ)) then
            Remove λ from Pop;
            Insert ρ in Pop;
        else
            Remove worst chromosome from Pop;
            Insert ρ;
        end
    else
        ++iter ;
    end
end
```

4.5 Experimental results

We tested our approach on the four UTC instances available: fall 2011 and 2012 and spring 2011 and 2012. The time limit used by the practitioner is usually a day. However, our approach takes generally much less than this time limit. The algorithm has been written in C++. The tests were run on a machine with an Intel Xeon E5-2670 and 32 GB of RAM and the compiler g++ (GCC) 4.4.7 was used. Each test was run 5 times and the one with the best score was selected.

4.5.1 Parameter Tuning

To be able to tune the parameters for memetic algorithm, a tuning was needed to be made on the local search operators. We varied a range of values on the set of parameters for each of the local search operators. All these parameters were varied using the default settings for the memetic approach: pop_size was set to 20 and $ITER\_MAX$ was set to n (n number of exams in the instance). p_m was set to 0.1.

We found that the following parameters for local searches produce overall the best results for the approach:

- ITER_MAX_HC was set 10.
- ITER_MAX_LDC was set to $10 * n$ and k was set to 20.
- ITER_MAX_SWAP was set to $10n$.

The default settings of the memetic algorithm are : 20 for pop_size; 5 for POP_OPT; 0.1 for p_m and n for ITER_MAX (n is the number of exams in the instance).

4.5.2 Analytical Results

The proposed approach was tested on the four available instances. Overall, the algorithm able to produce satisfying solutions and the solutions were better than all the solutions produced by the practitioner. However, as the approach contains different parameters, we aim at analyzing the impact of each the parameters of algorithm to determine the best configuration to adopt. The set of parameters are varied in ranges of values in order to extract the best value for each parameter. For each instance, the best solution in the population is taken as the representative for each test.

ITER_MAX is a key parameter of the approach. Increasing or decreasing it would influence the number of generations and the attempts of producing new solutions Figure 4.1 presents the results when varying $ITER\_MAX$ between n and $2n$. The columns in grey represent the results when ITER_MAX set to n and in black when the ITER_MAX is set to 2n. The figure shows that running the algorithm with $2n$ does not help find better solutions. Out of four instances, $2n$ is worse than n three times. This can be justified by the complexity of the problem

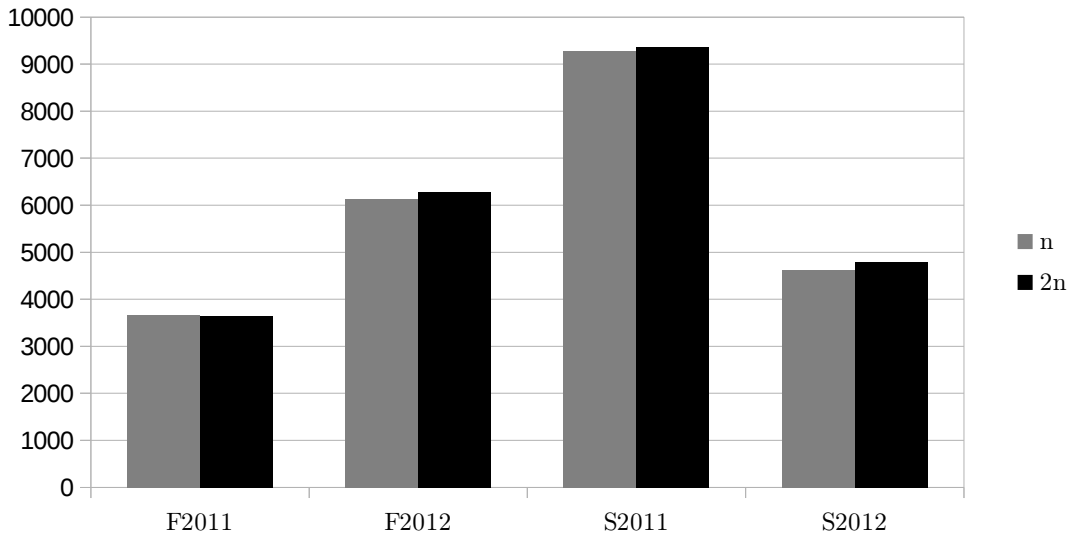


Figure 4.1 – Impact of ITER_MAX on the four instances

for the three instances and the existence of splittable and non-splittable exams. On the other hand, increasing the number of maximum iterations is not without consequence. The run time can increase with at least 10% when doubling the maximum number of iterations. This is in fact a draw back for some universities as they seek to have a timetable in the less time possible.

The probability of applying the local search can be a important factor in the algorithm. Table 4.2 shows the results on the four instances with the different values of p_m. We considered these values to test whether a high probability would improve the algorithm. Columns 0.1, 0.4, 0.5 and 0.6 present the different values of p_m that were tested. Each column contains two columns: the rum time (CPU) in seconds and the quality of the solution (Cost). The default value for p_m being 0.1, results show increasing the probability of calling the local search is not always beneficial. This can be explained by the fact that calling the local search more frequently does not necessary improve the best solution in the population. This shows that the impact of the probability on the best solution is usually not considerable as the most important part is the local search itself. It is also worth noting that increasing the probability does not impact the run time.

	0.1		0.4		0.5		0.6	
	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost
F2011	559.318	**3287**	535.903	3639	550.749	3319	516.099	3618
F2012	741.431	**6073**	775.871	6267	617.722	6226	836.289	6207
S2011	934.689	**9067**	1021.6	9318	1034.13	9500	984.456	9458
S2012	1517.66	**4525**	1437.61	4935	2001.5	4648	1437.32	4664

Table 4.2 – Impact of p_m on the four instances

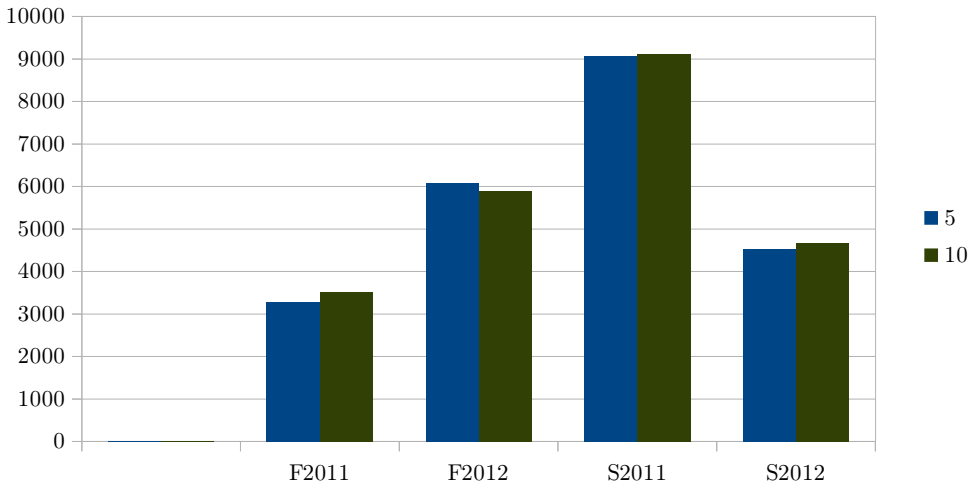


Figure 4.2 – Impact of POP_OPT on the four instances

In the population initialization, *POP_OPT* chromosomes are improved using the local search with the goal of creating good initial chromosomes for crossover later. To assess the impact of *POP_OPT*, we tested the algorithm by setting the number of improved chromosomes to 10. Figure 4.2 shows in blue the columns when the number of improved chromosomes is set to 5 and in dark green when it is set to 10.

The results show that, unlike what one would expect, increasing *POP_OPT* can worsen the results of the algorithm. It slightly improves the solution for F2012 but worsens it on the other three instances. We believe that the reason is that increasing the number of improved chromosome during initialization will decrease

the diversification the algorithm can get when crossing chromosomes.

The population size is without doubt of crucial importance when considering population-based algorithms. By default, our approach utilizes 20 chromosomes. However, we would like to know whether considering more chromosomes helps find better solutions by allowing multiple neighborhoods represented by the chromosomes.

Table 4.3 shows the results for the sizes of population 20, 30, 40 and 50. For each size, the run time (CPU) in seconds and the cost of the solution is reported. The test with the best results are in bold. The case of pop_size is not different from the one of POP_OPT. Results show that when the size of the population is increased, not only does the quality of the best solution decrease but the run time for each instance is generally higher. The size 20 seems to be the best comprise between all the others as, when it does not yield the best solution, the gap between its best solution and the best solution for other sizes is marginal.

	20		30		40		50	
	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost
F2011	559.318	**3287**	582.372	3463	682.105	3535	626.112	3431
F2012	741.431	**6073**	775.992	6391	858.259	6369	846.447	6481
S2011	934.689	**9067**	1142.35	9185	1181.56	9279	1271.59	9629
S2012	1517.66	**4525**	1571.57	4705	1556.34	4630	1597.16	4657

Table 4.3 – Impact of pop_size on the four instances

Finally, we compare the efficiency of the memetic algorithm with the approach used by the practitioner in Table 4.4. Column Best shows the best scores obtained by the memetic algorithm and column Average shows the average score for all the tests presented above. The results clearly show that our algorithm outperforms the solutions given by the practitioner. When these results were communicated to the practitioner, they expressed a good satisfaction about the timetable.

4.6 Conclusion

A memetic approach was presented for a practical timetabling problem. The problem is taken from the Université de Technologie de Compiègne (UTC). This practical problem consists of a mix of classical constraints that can be found in the

	Best	Average	Practitioner
F2011	3287	3798	7501
F2012	5891	6530	9266
S2011	9067	9711	10926
S2012	4493	4811	12008

Table 4.4 – Results of the proposed approach compared to the results of the approach used by the practitioner

benchmarks of the literature and a set of new constraints. The new constraints fall into the scope of the extensions proposed for the first academic problem.

In particular, the new constraints (hard and soft) consider exams that take place on multiple sites. The hard constraints forbids students from taking exams in two overlapping periods and rooms from being used more than once in two overlapping periods. Splittable exams are also managed using hard constraints. The new soft constraint imposes a penalty when a student has two exams scheduled in two periods in a row in the day day at two different sites.

The approach was tested on four available instances. A parameter tuning was performed to determine the best parameter settings for the approach. The tuning results showed that the probability of applying the local search and the size of the population does not have a big impact on the instances. The results also showed that the proposed approach outperforms the approach currently used by the practitioner. We aim at adapting our approach in the near future to the benchmarks of the literature and compare the results that will be obtained to existing ones. The four instances we used can also be used as additional instances to be added to the ITC2007 benchmark.

CHAPTER 5

MATHEMATICAL FORMULATION FOR THE STUDENT SCHEDULING PROBLEM

Contents

5.1 Introduction

University timetabling problems represent some of the problems that are periodically faced by universities each year. Within this category of problems, course and exam timetabling are the most encountered problems in literature. The Student Scheduling Problem (SSP) is included, whether implicitly as in the course timetabling or explicitly by considering it a sole problem. We refer the reader to Schaerf (1999) for detailed review on course and exam timetabling.

Timetabling problems are generally composed of soft and hard constraints. The hard constraints are to be respected in order to consider the solution feasible. The soft constraints are used to measure the quality of the solution using a penalty on the violation of each one of them.

For some institutions, students are grouped by year and must perform the same course curriculum. In such cases, dealing with SSP is much easier since groups of students have the same courses. In some other institutions, a set of timetables is made available for the students at the beginning of each term. Students have to choose some of their courses according to the rules that are stated by the institution. In some other cases, a preference list could be built by students. Despite the fact that SSP covers a large variety of real situations, the core problem remains assigning the maximum of students to their chosen courses while having for each assigned student an individual timetable that is conflict-free. Secondary objectives may also be encountered, for instance balancing the number of students in the groups of courses.

SSP is not as widely discussed as course and exam timetabling problems. Laporte and Desroches (1986) provided a mathematical model that considers respecting the list of preferences for the students as the only hard constraint. The conflicts between the sections are however considered as soft constraints and thus have to be minimized. The problem is solved in three phases: it first starts by finding a feasible solution, then balancing the sections and finally adjusting the solution to respect the room sizes. Cheng, Kruk, and Lipman (2003) referred to SSP as part of solving the American high school timetabling problem. Their goal was to respect the list of student preferences while having a conflict-free timetable. They demonstrate that SSP is an NP-hard problem and present a multi-commodity flow formulation to solve it. Broek, Hurkens, and Woeginger (2009) present the SSP in the TU Eindhoven. They provide two problem formulations and give complexity results on the different variants of the problem. For further solving approaches and discussions on SSP, we refer the reader to Tripathy (1992); Sabin and Winter (1986); Feldman and Golumbic (1989).

The remainder of the chapter is as follows. Section 5.2 presents a description of the student scheduling problem in the Université de Technologie de Compiègne and provides the different hard and soft constraints. Section 5.4 gives a formal mathematical model for the problem. Section 5.6 discusses the results obtained by

applying the model on the UTC instances and finally the conclusion is presented in Section 5.7.

5.2 Problem description

An academic year at Université de Technologie de Compiègne (UTC) is divided into two semesters of seventeen weeks. There are five engineering degrees decomposed into a two-year cycle called "fundamental studies" followed by a three-year cycle called "major studies". There exist a master program with four majors and fourteen specialization.

Eight departments share the same premises. Every semester, a timetabling of about three hundred Units of Value (UV) is built and there are in average two thousand and five hundred students. Timetables are built based on estimates of future student enrollments, with limited resources, and, dealing with constraints related to teachers and rooms. They were made available at each beginning of term. Students are required to choose at most seven UVs. A student of a department can choose a UV of another one if an individual conflict-free timetabling exists.

A UV is composed of activities that have to be performed by the student during the semester. Types of activities widely encountered are courses, tutorials lab activities, but other type of activities exist. As an example, students who choose the UV X have to be assigned into a weekly course activity, a weekly tutorial activity and a fortnightly lab activity.

Set of sections are scheduled weekly or fortnightly and correspond to activity. To illustrate with an example, a UV X corresponds to a unique section of a course with a 96 students capacity, 4 tutorial sections each with a 24 students capacity, and 8 fortnightly lab sections, each with a capacity of 12 students. There are in average a thousand and four hundred sections with the corresponding

At the beginning of the first week of the first term, students choose at most seven UVs while verifying on their own that a conflict-free timetable exists using the set of timetables. Timetables are updated to deal with real enrollments during the four first days of the first week of each semester. New sections are created and others are cancelled according to student enrollments and resource availabilities. Students may be rejected from some UVs by taking into account their individ-

ual curricula and alternative UVs are proposed. A team composed of heads of departments and timetabling makers take these decisions. When timetables and enrollments are finally stabilized, students are scheduled and they receive by e-mail their own individual timetable for the semester. A student is said to be fully scheduled when this student is assigned into a unique section for each activity of the chosen UVs. During the first week of the semester, it is of crucial importance for the team to have a tool at their disposal to take decisions. We need to detect students who choose a set of UVs such that there is no feasible timetable for all these UVs. These students are said pathological. In these cases, it is required for arbitration to know the maximum number of UVs where they can be assigned to and the list of these UVs among the chosen UVs.

5.3 Preprocessing

Before the data are processed by the the model, preprocessing them is needed to help reduce the problem's size. The idea is to detect infeasible sections in which students cannot be assigned to remove them.

Consider a set of UVs, each of which has activities. Activities correspond to a set of sections with events planned in the timetables. A section is mandatory for an activity if it is unique. The idea is to eliminate sections which have time slots that intersect with those of mandatory sections.

For example, consider the following timetable:

	8h00 - 10h00		10h15 - 12h15		14h15 - 16h15
monday	UV01 C		UV02 D2		UV01 D1
	UV02 D1		UV03 C		UV02 C
	UV03 D1				
thuesday	UV04 C		UV01 D2		
			UV04 D1		UV04 D2

Letter C stands for course and D for training. There are four UV, UV01, UV02, UV03 and UV04. Each UV has two activities: course and training. For instance,

UV01 has one course section (UV01 C) and two training sections (UV01 D1 and UV01 D2).

Assume next three students, E1 chooses (UV01, UV02, UV03), E2 chooses (UV02, UV03), and E3 chooses (UV01, UV02). It is obvious that student E1 cannot be scheduled: there is an intersection between the time slot of UV01 C course section and the time slot of the unique training section UV03 D1. Consider next student E2, the UV03 D1 unique training section eliminates the UV02 D1 training section since there is an intersection, and, the UV03 C course section eliminates the UV02 D2 training section since there is an intersection. Student E2 cannot be scheduled. Consider student E3, the UV01 C course section eliminates the UV02 D1 training section, the UV02 C section eliminates the UV01 D1 training section. So for student E3:

	8h00 - 10h00		10h15 - 12h15		14h15 - 16h15
monday	UV01 C		UV02 D2		
					UV02 C
thuesday			UV01 D2		

is the unique feasible timetable.

The students that cannot be scheduled are called pathological. The reason can come from their bad initial choices for their UV – without checking if a feasible individual timetable exists within the set of timetables made available at the beginning of term–, or from the modifications of the timetables make them pathological. They must be detected and treated apart.

Students with their chosen UV are considered in turn. The elimination procedure is repeatedly performed until no new unique section appears. The algorithm stops when a pathological case occurs.

The algorithm takes $\mathcal{L}$ a list of UV and returns $\mathcal{LL}_{sec}$ the list of list of sections for a list of UV. If $\mathcal{LL}_{sec}$ is empty, we cannot find a feasible timetable for the list of UV: student is pathological. Otherwise, since unique sections eliminate some other sections, we can obtain a reduced number of section for each activity for each UV. For the sake of simplicity, we do not detail functions we used in the algorithm.

Algorithm 8: filter_list_chosen_UV($\mathcal{L}$)

Data: $\mathcal{L}$ list of UV
Result: $\mathcal{LL}_{sec}$ list of lists of sections
$\mathcal{LL}_{sec} \leftarrow$ BuildListsOfSectionsFromActivityOfEachUV($\mathcal{L}$);
/* L_u is the list of unique sections */
$\mathcal{L}_u \leftarrow$ BuildListsOfUniqueSections($\mathcal{LL}_{sec}$);
$B\_stop \leftarrow 0$;
if *empty*(L_u) **then**
  $B\_stop \leftarrow 1$
end
while $B\_stop = 0$ **do**
  $sec \leftarrow pop(L_u)$;
  $\mathcal{LL}_{sec} \leftarrow$ EliminateSectionsThatIntersect($\mathcal{LL}_{sec}$,sec);
  $L_u \leftarrow$ FindNewUniqueSectionAndUpdate($\mathcal{LL}_{sec}$,L_u) ;
  if *empty*(L_u) **then**
    $B\_stop \leftarrow 1$;
  end
  if *NoSectionForAnActivity*($\mathcal{LL}_{sec}$) = *1* **then**
    $B\_stop \leftarrow 1$;
    $\mathcal{LL}_{sec} \leftarrow \emptyset$
  end
end
return $\mathcal{LL}_s$

5.4 Mathematical Model

We present in this section a mathematical model for the student scheduling problem. The objective is to maximize the number of the students totally assigned to their activities. A student is totally assigned if they are assigned to one section of all the activities. list of preference. We propose a set of valid inequalities aiming at accelerating the model.

5.4.1 Sets, Parameters and variables

Set definitions:

$\mathcal{S}$: set of students, indices associated with students are s and s';

$\mathcal{A}$: set of activities, indices associated with activities are a and a';

$\mathcal{K}$: set of sections, indices associated with sections are k and k';

$\mathcal{A}_s$: set of activities chosen by student s deduced from list of chosen UVs;

$\mathcal{K}_a$: set of sections of activity a;

$\mathcal{F}_{aa'}$: set of couples $[k, k']$, such that $k \neq k'$, of two sections k and k' of two activities a and a'. At most one of the sections k or k' should be used, and hence students are assigned either into k or k';

$m_{kk'}$: Boolean, one if there is a conflict between section k and section k', $k \neq k'$, zero otherwise. Two sections are in conflict if their time slots intersects or if they belong to the same activity.

p_k : positive integer, standard maximum size of a section k.

Decision Variables

T_s : Boolean, set to one if student s is assigned into a unique section for each activity in $\mathcal{A}_s$, zero otherwise;

Y_{sa} : Boolean, set to one if student s is assigned into a unique section of activity a, zero otherwise;

Z_{sak} : Boolean, set to one if student s is assigned into section k of activity a, zero otherwise;

5.4.2 The Formulation

The objective is to maximize the number of students that are totally assigned to all courses in their preference list. That is to say the number of students s assigned into a unique section for each activity in $\mathcal{A}_s$.

Maximize:

$$\sum_{s \in \mathcal{S}} T_s \tag{5.1}$$

subject to:

$$\forall s \in \mathcal{S} \qquad \sum_{a \in \mathcal{A}_s} Y_{sa} \geq |\mathcal{A}_s| T_s \tag{5.2}$$

$$\forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}_s \qquad Y_{sa} = \sum_{k \in \mathcal{K}_a} Z_{sak} \tag{5.3}$$

$$\forall a \in \mathcal{A} \quad \forall k \in \mathcal{K}_a \qquad \sum_{s \in \mathcal{S}} Z_{sak} \leq p_k \tag{5.4}$$

$$\forall s \in \mathcal{S} \quad \forall a, a' \in \mathcal{A}_s \quad \forall k \in \mathcal{K}_a \quad \forall k' \in \mathcal{K}_{a'} \quad \text{such that } m_{kk'} = 1 \qquad Z_{sak} + Z_{sa'k'} \leq 1 \tag{5.5}$$

$$\forall s \in S \quad \forall a \in \mathcal{A}_s \quad \forall s' \in \mathcal{S} \quad \forall a' \in \mathcal{A}_{s'} \quad \forall [k, k'] \in \mathcal{F}_{aa'} \qquad Z_{sak} + Z_{s'a'k'} \leq 1 \tag{5.6}$$

$$T_s, Y_{sa}, Z_{sak} \in \{0, 1\} \tag{5.7}$$

Let us consider Equation (5.1). At the optimum, we obtain N_t the maximum number of students s assigned into a unique section for each chosen activity $\mathcal{A}_s$. Equations (5.2) set $T_s = 1$ *iff* student s is assigned into a unique section for each chosen activity. The quantity $|\mathcal{A}_s|$ equals the number of chosen activities for a student s. Equations (5.3) link the decision variables Y_{sa} and Z_{sak}. The section capacities are always respected using Equations (5.4). Conflicts between any two sections k and k' such that $m_{kk'} = 1$ are enforced using Equations (5.5). Equations (5.6) ensure that two students s and s' cannot be assigned into sections k or k' at the same time, so, at most one of the sections k or k' is used. We define as M_{TA} the set of Equations (5.1) to (5.7).

5.5 Valid inequalities

We propose valid inequalities to help speeding up the computing time.

We first used the classical clique inequalities. The cliques used to build valid inequalities are extracted from the conflict graph for a student. Nodes are sections $k \in \left(\bigcup_{a \in \mathcal{A}_s} \mathcal{K}_a\right)$ and an edge $[k, k']$ between two nodes k and k' corresponds to a conflict. We compute the set of maximum cliques $\mathcal{C}_s$. For every clique $c \in \mathcal{C}_s$, the students can attend at most one section. Hence, we have the following valid

inequality:

$$\forall s \in \mathcal{S} \quad \forall c \in \mathcal{C}_s \qquad \sum_{k \in c} Z_{sak} \leq 1 \tag{5.8}$$

When linking the variables T_s and Y_{sa} in Equations (5.2), the following inequality is implicit:

$$\forall s \in \mathcal{S} \quad \sum_{a \in A_s} Y_{sa} \leq (\mid A \mid -1) + T_s \tag{5.9}$$

Its aim is to force T_s to be equal to $\mid A \mid$ whenever all the variables Y_{sa} of a student s are set to one. Since we have a maximization problem, if all the variables Y_{sa} are set to one, this inequality is implicit since T_s would be set to $\mid A \mid$ automatically. We would like to add this inequality to measure its impact on the run times of the model.

When a student is assigned to an activity of a UV, they should be assigned to all the other activities of the same UV. Hence, for two activities a and a' from the same UV u, we consider the following constraints:

$$\forall u \in \mathcal{U} \quad \forall a, a' \in \mathcal{A}_u \quad a \neq a' \qquad \sum_{s \in (\mathcal{S}_{ua} \cap \mathcal{S}_{ua'}),\ k \in \mathcal{K}_a} Z_{sak} = \sum_{s \in (\mathcal{S}_{ua} \cap \mathcal{S}_{ua'}),\ k' \in \mathcal{K}_{a'}} Z_{sa'k'} \tag{5.10}$$

The fourth type of valid inequalities involves an order between the variables. It is known in integer programming that imposing an order on variables can drastically speed up the model because it breaks the symmetry within the problem. The variable T_s, equals one if the student s is fully assigned, bit equals zero if student s is not assigned to at least one activity. Therefore, we obtain the following valid inequality:

$$\forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}_s \qquad T_s \leq Y_{sa} \tag{5.11}$$

Note that Equations (5.11) is the non-compact form of Equations (5.2).

5.6 Results

We tested the model M_{TA} on the set of instances taken from the real-world data from UTC. We started by modeling the problem with the practitioner in order to

Instance	S	U	A	G	Density	RealS	NbVar	NbConst
F10	2268	280	569	1396	0.46	2242	159948	817707
F11	2425	290	577	1435	0.41	2406	175197	962847
F12	2358	300	583	1446	0.43	2322	155644	797774
F13	2365	310	608	1466	0.44	2344	156936	785628
S10	2137	305	619	1409	0.39	2137	144931	823068
S11	2350	303	614	1415	0.45	2329	157443	902394
S12	2471	305	603	1467	0.38	2437	164951	989417
S13	2434	311	609	1488	0.42	2396	154124	866842
S14	2317	303	606	1417	0.43	2276	138606	667172

Table 5.1 – Characteristics of the instances

obtain the dataset. Next and after several meetings, we obtained nine instances corresponding to semesters from year 2010 to year 2014. Our model was implemented in C++ using CPLEX 12.5 as a solver and G++ 4.7 as a compiler. The tests were run on a machine with an Intel core i7-950 CPU 3.07 GHz and 24 GB of RAM.

Table 5.1 shows the characteristics of the different fall (F) and spring (S) semesters in the university. Column *S* gives the number of the students to schedule for the corresponding semester. Column *U* shows the number of UVs that can be chosen by the students. Column *A* presents the total number of activities of all UVs and column *G* the total number of sections of all the UVs that the students will take. Finally, column *Density* shows the density of the graph build between the UVs. Nodes correspond to the UVs and an edge is added between two UVs if they have students in common.

A first glimpse at Table 5.1 shows that the fall semesters (as well as spring semesters) are similar. The number of students in the university didn't rise since a few years and so is the case for the UVs. The density of the graph, however, indicates that there is a considerable interaction between the UVs.

To assess the impact of the cliques cuts on the instances, cliques are computed on the graphs of students. Students made different choices and have therefore different graphs on which cliques are computed. For each student, we computed the set of maximal cliques between the different sections.

Table 5.2 shows the different sizes and numbers of cliques per student. Col-

Instance	Total	MaxNb	AverageNb	MinNb	MaxSize	MinSize
F10	72247	113	32.2244	0	8	2
F11	76535	125	31.8101	0	6	2
F12	64585	95	27.8144	1	11	2
F13	65917	95	28.1216	0	7	2
S10	58541	88	27.394	0	7	2
S11	64569	88	27.7239	0	3	2
S12	67274	100	27.6053	0	37	2
S13	61653	90	25.7316	0	35	2
S14	56685	88	24.9055	0	30	2

Table 5.2 – Number and size of maximal cliques per student

umn Total shows the total number of cliques extracted for all the students of the instance. Column MaxNb presents the maximum number of cliques for a single student whereas column AverageNb shows the average number of cliques per student. Column MinNb gives the minimum number of cliques per student and Columns MaxSize and MinSize show the size of the maximum clique and the size of the smallest maximal clique respectively. Results on MinNb show that certain students do not have any cliques. This can signify that these students can be easy to assign and a possible policy is to schedule them at the end of the schedule.

We run the model on the different instances and obtained optimality for all the instances. Table 5.3 presents the results of the model with and without preprocessing. Column Heuristic shows the number of non-assigned students using the heuristic. It takes about five minutes on another computer. This heuristic has been coded on a very special environment and it would have been tedious to recode to get run times. Column MIP reports the optimal number of non-assigned students with the proposed model. Column "Default Settings" represent the tests in which the model was run without preprocessing and valid inequalities. Column "With preprocessing" gives the results when the preprocessing is activated. For these two columns, we provide NbNodes the number of nodes visited by the solver and CPU the run time in seconds.

The table shows that our model is effective on the different instances and that it reaches optimality for all instances in less than five minutes. In six cases over ten, all the students are assigned. Note that the non-assigned students are treated

			Default settings		With preprocessing	
instance	Heuristic	MIP	NbNodes	CPU(s)	NbNodes	CPU(s)
a2010	9	0	0	18.41	0	8.19
p2010	8	0	0	9.71	0	8.21
a2011	15	0	0	22.99	0	22.57
p2011	14	3	443	42.09	440	48.37
a2012	19	9	0	12.65	0	8.75
p2012	10	0	0	15.17	0	16.83
a2013	21	11	867	133.36	42	30.61
p2013	17	0	0	12.81	0	5.21
a2014	23	6	2016	206.22	140	24.82
p2014	12	0	0	8.87	0	5.22

Table 5.3 – Impact of the preprocessing on the run time and the number of developed nodes

by hand which takes a very long time. Moreover, we can manage limited resources while the former heuristic cannot.

Results clearly show that the preprocessing has a big influence on the run times of the formulation. This is justified by the reduction of the size of the problem done by the preprocessing using the eliminating of groups for students.

The different valid inequalities proposed are tested on the formulation to determine their efficiency. Table 5.4 presents comparative results of the different valid inequalities. To clearly assess the effect of the maximal clique cuts, we chose to deactivate the clique cuts of the solver.

Column "Without CPLEX cliques" refers to the default settings run with the CPLEX solver cuts deactivated. Column "Maximal cliques" gives the results of the model when all maximal cliques are added to the model while deactivating the solvers' clique cuts. Column $Y_{sa} <=| As | -1 + T_s$ presents the results obtained by adding the second type of valid inequalities to the "Default settings" model. Columns $\sum Z_{sak} = \sum Z_{sa'k'}$ and $T_s <= Y_{sa}$ corresponds the results obtained by adding the third and the fourth type of valid inequalities to the "Default settings" model respectively.

Table 5.4 shows that the solver's clique cuts can be ineffective for some in-

	Default settings		Without CPLEX cliques		Maximal cliques	
instance	NbNodes	CPU(s)	NbNodes	CPU(s)	NbNodes	CPU(s)
a2010	0	18.41	0	18.56	0	18.57
p2010	0	9.71	0	9.72	0	11.18
a2011	0	22.99	0	24.01	0	17.42
p2011	443	42.09	528	88.29	528	222.36
a2012	0	12.65	310	27.81	310	29.93
p2012	0	15.17	0	22.95	0	16.36
a2013	867	133.36	442	103.69	442	203.01
p2013	0	12.81	0	9.41	0	7.81
a2014	2016	206.22	504	110.73	504	154.69
p2014	0	8.87	0	10.17	0	6.91

	$\sum Z_{sak} = \sum Z_{sa'k'}$		$Y_{sa} <= \mid As \mid -1 + T_s$		$T_s <= Y_{sa}$	
	NbNodes	CPU(s)	NbNodes	TTotal	NbNodes	CPU(s)
a2010	0	18.21	0	16.28	0	13.22
p2010	0	10.75	0	10.74	0	7.8
a2011	0	66.09	0	22.65	0	15.29
p2011	1117	122.49	398	45.66	0	10.24
a2012	0	20.96	0	13.31	0	8.52
p2012	0	21.48	0	17.42	0	10.64
a2013	511	156.86	155	48.47	0	10.11
p2013	0	12.9	0	11	0	7.65
a2014	1010	168.49	2068	103.28	0	8.04
p2014	0	10.25	0	10.33	0	6.07

Table 5.4 – Impact of the different valid inequalities on the instances

stances because removing them yields better results. The maximal cliques extracted from students graphs are however mostly not helpful as it is the case with instance p2011. We believe that the reason comes from the considerable number of cliques for each instance. An improvement would be reducing the number of cliques considered and attempting to consider the "useful" cliques for each student.

The valid inequality $\sum Z_{sak} = \sum Z_{sa'k'}$ presents an interesting case. The run times and the number of nodes explored increase for certain instances (p2011) and decrease for others (a2014). The third type of inequalities does not considerably impact the results. As expected, these valid inequalities are implicit and in most

cases not beneficial. However, they might present a good help to tighten the bounds in the case of hard instances.

The fourth type of valid inequalities are the most effective inequalities. Their results improve the results of all instances and present the shortest run time among all the tests. We believe the effectiveness of these valid inequalities is a result of the order imposed on the variables which helps the solver quickly take decisions.

5.7 Conclusion

This chapter presented preprocessing and mathematical models for the practical student scheduling problem taken from Université de Technologie de Compiègne. The preprocessing presented a procedure that eliminates sections that are simultaneous to other mandatory sections and detects pathological students for which it is impossible to find a feasible timetable. The aim is to maximize the number of totally assigned students. To accelerate the model, four types of valid inequalities were proposed. Out of the four valid inequalities, three were effective and gave better results than the model alone. The preprocessing and the valid inequalities proved to be effective by halving the run time and providing solution within less than five minutes. The result presented also showed that we were able to solve the model to optimality within a very short time.

Conclusion and Future Work

In this thesis, we presented different methodologies and strategies for university timetabling problems. Mathematical models, lower bounds and a memetic approach were used to solve both academic and practical problems. The exam timetabling and student scheduling problems constitute the most studied subproblems and were covered in this thesis.

After introducing the optimization problems related to timetabling in Chapter 1, we present in Chapter 2 an improved mathematical model for the exam timetabling as described in the third track of the second International Timetabling Competition (ITC2007). We presented a preprocessing procedure that helps reveal hidden constraints between exams based on the original ones. We also improved an existing model that was used to evaluate solutions and made it possible to find feasible solutions for more instances. To reduce the run time of the improved model, we proposed valid inequalities that were derived from the well-known Bin Packing dual-feasible functions. The results showed that the improved model reaches optimality for more instances and uses less memory than the original one.

Chapter 3 investigates lower bound techniques, a more compact reformulation and a constraint programming model for the exam timetabling problem. Lower bounds are assessed using a set covering model applied on the cliques that are extracted from the conflict graph between the exams. We propose next a reformulation of the constraints to reduce their number from n^2 to n. Moreover, we show that the constraint programming model succeeds in finding better results compared to the linear model within the time limit.

To tackle the practical timetabling problem of Université de Technologie de Compiègne, we proposed in Chapter 4 a memetic approach with three different local search operators. The solutions are represented as chromosomes using the indirect incoding and are stored in the population. The local search opera-

tors compromise Hill Climbing, Swap and Light Destruction/Construction. We showed that the proposed algorithm outperforms the existing approach used in the university and proposed to extend the set of instances of the ITC2007 collection.

For the student scheduling problem, we discussed in Chapter 5 the problem encountered at Université de Technologie de Compiègne. We proposed a preprocessing stage that reduces the size of the students' timetables by removing the time slots in which they cannot be scheduled in. The preprocessing stage also allows us to detect infeasible timetables for students who wrongly choose their courses. Next, the mathematical model is presented. The objective aims at maximizing the number of students that have a timetable which totally satisfies their demand. Moreover, we proposed a set of valid inequalities to accelerate the model run time. The results show that the preprocessing reduces the problem's size by reducing the number of variables and constraints relative to each student. The model we proposed is able, with the help of the valid inequalities, to reach optimality within a very short time.

Future Work

The contributions in this thesis clear the way for numerous future work to be done on university timetabling. We provide in the following the ongoing work as well as the future openings we plan to investigate.

Combined Framework. Course and exam timetabling are closely related problems which, when considered together in a fully automated system, will provide a comprehensive solution to an academic institution. Investigating a combined framework has the potential to provide the necessary search engine in a multilayer system that could solve both the course and exam timetabling problems. For such a framework, advanced heuristic and metaheuristics are needed. hyperheuristics and heuristics combined with mathematical programming can be a promising path.

Constraint Programming. The constraint programming model that we pro-

vided in Chapter 3 presents an encouraging beginning to the application of constraint programming to university timetabling. The ease and the flexibility of implementation and adaptation offered by constraint programming encourage us to apply and adapt the same approach for the student scheduling problem and course timetabling. It is also in our ongoing work to enhance the current model by trying to add cuts and inequalities and propose some propagation constraints in order to help the model provide better results.

A Benchmark For The Student Scheduling Problem. The contribution proposed for the student scheduling problem encourages us to investigate providing the community with a set of instances for the problem. Currently, the lack of a common testing base makes it difficult to compare the different solution methods proposed. To that end, the set of instances will be made online for the researchers.

Column Generation For University Timetabling. The difficulty encountered using the mixed integer programming model we proposed in Chapter 2 incites us to investigate a column generation approach for the exam timetabling problem. A possible start would be trying the generic column generation solvers to check whether the approach is suitable for the problem. The adaptation of the column generation approaches applied to course timetabling can also help reach this goal.

BIBLIOGRAPHY

Abdullah, S., E. K. Burke, and B. Mccollum. 2005. "An investigation of variable neighbourhood search for university course timetabling". In: *The 2nd multidisciplinary international conference on scheduling: theory and applications (MISTA)*, pp. 413–427.

Abdullah, S., E. K. Burke, and B. McCollum. 2007. "Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem". In: *Proceedings of the sixth International Conference of Metaheuristics*, pp. 153–169.

Abdullah, S., H. Turabieh, B. McCollum, and P. McMullan. 2010. "A Tabu-Based Memetic Approach for Examination Timetabling Problems". In: *Proceedings of the 5th International Conference of Rough Set and Knowledge Technology RSKT 2010, Beijing, China, October 15-17*. Vol. 6401. Lecture Notes in Computer Science, pp. 574–581.

Arbaoui, T., J.-P. Boufflet, and A. Moukrim. 2013. "An analysis framework for examination timetabling". In: *Proceedings of the Sixth International Symposium on Combinatorial Search (SoCS 2013)*. Leavenworth, WA, USA, pp. 11–19.

Atsuta, M., K. Nonobe, and T. Ibaraki. 2008. "ITC2007 track 1: An Approach using General CSP Solver". In: *Proceedings of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

Blum, C. and A. Roli. 2003. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM Computing Survey* 35.3, pp. 268–308.

Bonabeau, E., M. Dorigo, and G. Theraulaz. 1999. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc.

Boufflet, J. P. and S. Nègre. 1995. "Three Methods Used to Solve an Examination Timetable Problem". In: *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, pp. 327–344.

Broek, J. Van den, C. Hurkens, and G. Woeginger. 2009. "Timetabling problems at the TU Eindhoven". In: *European Journal of Operational Research* 196.3, pp. 877–885.

Burke, E., K. Jackson, J. Kingston, and R. Weare. 1997a. "Automated University Timetabling: The State of the Art". In: *The Computer Journal* 40.9, pp. 565–571.

Burke, E., Y. Bykov, J. Newall, and S. Petrovic. 2004a. "A Time-Predefined Local Search Approach to Exam Timetabling Problems". In: *IIE Transactions on Operations Engineering* 36.6, pp. 509–528.

Burke, E. K., G. Kendall, and E. Soubeiga. 2003. "A tabu-search hyperheuristic for timetabling and rostering". In: *Journal of Heuristics* 9.6, pp. 451–470.

Burke, E. K. and J. P. Newall. 2004. "Solving Examination Timetabling Problems through Adaption of Heuristic Orderings". In: *Annals of Operations Research* 129.1-4, pp. 107–134.

Burke, E. K., J. P. Newall, and R. F. Weare. 1996. "A memetic algorithm for university exam timetabling". In: *Proceedings of the first International Conference on Practice and Theory of Automated Timetabling, Edinburgh, U.K., August 29 - September 1, 1995*. Vol. 1153. Lecture Notes in Computer Science, pp. 241–250.

Burke, E. K., D. Elliman, P. H. Ford, and R. F. Weare. 1995. "Examination timetabling in british universities: A Survey". In: *Proceedings of the First International Conference of Practice and Theory of Automated Timetabling, Edinburgh, U.K., August 29 - September 1, 1995*. Vol. 1153. Lecture Notes in Computer Science, pp. 76–90.

Burke, E. K., K. Jackson, J. H. Kingston, and R. Weare. 1997b. "Automated university timetabling: The state of the art". In: *The computer journal* 40.9, pp. 565–571.

Burke, E. K., G. Kendall, M. Mısır, E. Özcan, E. Burke, G Kendall, E Özcan, and M Mısır. 2004b. *Applications to timetabling*. Ed. by J. L. Gross and J. Yellen.

Burke, E. K., B. McCollum, A. Meisels, S. Petrovic, and R. Qu. 2007. "A Graph-Based Hyper-Heuristic for Educational Timetabling Problems". In: *European Journal of Operational Research* 176.1, pp. 177–192.

Burke, E. K., B. McCollum, P. McMullan, and A. J. Parkes. 2008. "Multi-objective aspects of the examination timetabling competition track". In: *Proc. of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

Carlier, J., F. Clautiaux, and A. Moukrim. 2007. "New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation". In: *Computers & Operations Research* 34.8, pp. 2223–2250.

Carter, M. W. and G. Laporte. 1998. "Recent developments in practical course timetabling". In: *Proceedings of the Second International Conference of Practice and Theory of Automated Timetabling*. Vol. 1408. Lecture Notes in Computer Science, pp. 3–19.

Carter, M. W., G. Laporte, and S. Y. Lee. 1996. "Examination timetabling: Algorithmic strategies and applications". In: *Journal of the Operational Research Society* 47.3, pp. 373–383.

Černý, V. 1985. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of Optimization Theory and Applications* 45.1, pp. 41–51.

Cheng, E., S. Kruk, and M. Lipman. 2003. "Flow formulations for the student scheduling problem". In: *Proceedings of the Fourth International Conference of Practice and Theory of Automated Timetabling*. Vol. 2740. Lecture Notes in Computer Science, pp. 299–309.

Clausen, J. 1997. "Branch and bound algorithms: principles and examples". In: *Parallel Computing in Optimization*, pp. 239–267.

Clements, D. P. and D. Joslin. 1999. "Squeaky Wheel Optimization". In: *Journal of Artificial Intelligence Research* 10, pp. 353–370.

Colorni, A., M. Dorigo, and V. Maniezzo. 1992. "Distributed Optimization by Ant Colonies". In: *Proceedings of the First European Conference on Artificial Life*, pp. 134–142.

Corne, D., M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price. 1999. "New ideas in optimization". In: McGraw-Hill Ltd., UK.

Costa, D. 1994. "A tabu search algorithm for computing an operational timetable". In: *European Journal of Operational Research* 76.1, pp. 98–110.

Dantzig, G. and M. Thapa. 2003. *Linear Programming: Theory and Extensions*. Linear Programming: Theory and Extensions.

De Smet, G. 2008. "ITC 2007 – Examination Track". In: *Proceedings of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

Demeester, P., B. Bilgin, P. Causmaecker, and G. Berghe. 2012. "A Hyperheuristic Approach to Examination Timetabling Problems: Benchmarks and a New Problem from Practice". In: *Journal of Scheduling* 15.1, pp. 83–103.

Desaulniers, G., J. Desrosiers, and M. M. Solomon, eds. 2005. *Column Generation*.

Dueck, G. 1993. "New Optimization Heuristics: the Great Deluge Algorithm and the Record-to-Record Travel". In: *Journal of computational physics* 104.1, pp. 86–92.

Feldman, R. and M. C. Golumbic. 1989. "Constraint satisfiability algorithms for interactive student scheduling". In: *Proceedings of the 11th international joint conference on Artificial intelligence*. Vol. 2, pp. 1010–1016.

Feo, T. A. and M. G. C. Resende. 1995. "Greedy Randomized Adaptive Search Procedures". In: *Journal of Global Optimization* 6, pp. 109–133.

Fonseca, G. H. G. and H. G. Santos. 2013. "A New Integer Linear Programming Formulation to the Examination Timetabling Problem". In: *The 6th Multidisciplinary International Conference on Scheduling: Theory and Applications (Mista 2013)*. Gent, Belgium, pp. 345–355.

Garey, M. R. and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Glover, F. and G. Kochenberger, eds. 2003. *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Glover, F. and M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers.

Glover, F. 1975. "Improved linear integer programming formulations of nonlinear integer problems". In: *Management Science* 22.4, pp. 455–460.

Gogos, C., P. Alefragis, and E. Housos. 2012. "An Improved Multi-staged Algorithmic Process for the Solution of the Examination Timetabling Problem". In: *Annals of Operations Research* 194 (1), pp. 203–221.

Hamilton-Bryce, R., P. McMullan, and B. McCollum. 2014. "Directed Selection using Reinforcement Learning for the Examination Timetabling Problem". In: *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, York, UK.* Pp. 218–232.

Hansen, P. and N. Mladenovi. 2001. "Variable neighborhood search: Principles and applications". In: *European Journal of Operational Research* 130.3, pp. 449–467.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hoos, H. H. and T. Stutzle. 2004. *Stochastic Local Search : Foundations & Applications (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann.

IBM. 2012. *CPLEX User's Manual*. URL: `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5`.

Karp, R. M. 1972. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*, pp. 85–103.

Kennedy, J. and R. C. Eberhart. 1995. "Particle Swarm Optimization". In: *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948.

Khachiyan, L. G. 1979. "A polynomial algorithm for linear programming (in Russian)". In: *Proceedings of the USSR Academy of Sciences*. 244, pp. 1093–1096.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. "Optimization by simulated annealing". In: *Science* 220, pp. 671–680.

Kostuch, P. 2005. "The University Course Timetabling Problem with a Three-phase Approach". In: *Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling*. PATAT'04, pp. 109–125.

Laporte, G. and S. Desroches. 1986. "The problem of assigning students to course sections in a large engineering school". In: *Computers & operations research* 13.4, pp. 387–394.

Leung, J. Y. 2004. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC Press.

McCollum, B. and N Ireland. 2006. "University timetabling: Bridging the gap between research and practice". In: *E Burke, HR, ed.: PATAT*, pp. 15–35.

McCollum, B., P. McMullan, E. K. Burke, A. J. Parkes, and R. Qu. 2007. *The Second International Timetabling Competition: Examination Timetabling Track*. Tech. rep. QUB/IEEE/TECH/ITC2007/Exam/v4.0. Queen's University, Belfast.

McCollum, B., P. McMullan, A. Parkes, E. K. Burke, and S. Abdullah. 2009. "An Extended Great Deluge Approach to the Examination Timetabling Problem". In: *Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications 2009 (MISTA 2009)*. Dublin, Ireland, pp. 424–434.

McCollum, B., A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. 2010a. "Setting the research agenda in

automated timetabling: The second international timetabling competition". In: *INFORMS Journal on Computing* 22.1, pp. 120–130.

McCollum, B., A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke. 2010b. "Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition". In: *INFORMS Journal on Computing* 22.1, pp. 120–130.

McCollum, B., P. McMullan, A. Parkes, E. K. Burke, and R. Qu. 2012. "A New Model for Automated Examination Timetabling". In: *Annals of Operations Research* 194 (1), pp. 291–315.

Meignan, D., A. Koukam, and J.-C. Créput. 2010. "Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism". In: *Journal of Heuristics* 16, pp. 859–879.

Merlot, L. T., N. Boland, B. D. Hughes, and P. J. Stuckey. 2003. "A hybrid algorithm for the examination timetabling problem". In: *Lecture notes in computer science: Practice and theory of automated timetabling IV: selected papers from the 4th international conference*. Vol. 2740. Springer, pp. 207–231.

Metaheuristics Network. 2002. URL: `http://www.idsia.ch/Files/ttcomp2002`.

Moscato, P. 1989. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms*. Tech. rep. Caltech Concurrent Computation Program.

Müller, T. 2009. "ITC2007 Solver Description: a Hybrid Approach". In: *Annals of Operations Research* 172 (1), pp. 429–446.

Östergård, P. R. 2002. "A Fast Algorithm for the Maximum Clique Problem". In: *Discrete Applied Mathematics* 120.1–3, pp. 197 –207.

Papadimitriou, C. H. and K. Steiglitz. 1998. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications.

Pardalos, P. M. and M. G. C. Resende, eds. 2002. *Handbook of Applied Optimization*. Oxford University Press, USA.

Park, K. and B. Carter. 1995. "On the effectiveness of genetic search in combinatorial optimization". In: *Proceedings of the 10th ACM Symposium on Applied Computing*, pp. 329–336.

Parkes, A. J. and E. Özcan. 2010. "Properties of Yeditepe examination timetabling benchmark instances". In: *Proc. of the 8th international conference on the practice and theory of automated timetabling (PATAT 2010)*.

Pillay, N. 2008. "A Developmental Approach to the Examination Timetabling Problem". In: *Proc. of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

Qu, R., E. K. Burke, B. McCollum, L. T. Merlot, and S. Y. Lee. 2009a. "A survey of search methodologies and automated system development for examination timetabling". In: *Journal of scheduling* 12.1, pp. 55–89.

Qu, R., E. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee. 2009b. "A Survey of Search Methodologies and Automated System Development for Examination Timetabling". In: *Journal of Scheduling* 12.1, pp. 55–89.

Rahman, S. A., A. Bargiela, E. K. Burke, E. Özcan, B. McCollum, and P. McMullan. 2014. "Adaptive linear combination of heuristic orderings in constructing examination timetables". In: *European Journal of Operational Research* 232.2, pp. 287 –297.

Roli, A. and M. Milano. 2002. "MAGMA: A Multiagent Architecture for Metaheuristics". In: *IEEE Trans. on Systems, Man and Cybernetics - Part B* 34, pp. 925–941.

Sabar, N. R., M. Ayob, R. Qu, and G. Kendall. 2012. "A Graph Coloring Constructive Hyper-Heuristic for Examination Timetabling Problems". In: *Applied Intelligence* 37.1, pp. 1–11.

Sabin, G. and G. Winter. 1986. "The impact of automated timetabling on universities-a case study". In: *Journal of the operational Research Society* 37.7, pp. 689–693.

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

Schrimpf, G., J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. 2000. "Record Breaking Optimization Results Using the Ruin and Recreate Principle". In: *Journal of Computational Physics* 159.2, pp. 139–171.

Socha, K., J. Knowles, and M. Sampels. 2002. "A MAX-MIN Ant System for the University Course Timetabling Problem". In: *in Proceedings of the 3rd International Workshop on Ant Algorithm, ANTS 2002, Lecture Notes in Computer Science*. Springer-Verlag, pp. 1–13.

Soghier, A. and R. Qu. 2013. "Adaptive selection of heuristics for assigning time slots and rooms in exam timetables". In: *Applied Intelligence*, pp. 1–13.

Tripathy, A. 1992. "Computerised decision aid for timetabling—a case analysis". In: *Discrete applied mathematics* 35.3, pp. 313–323.

Turabieh, H. and S. Abdullah. 2011. "An Integrated Hybrid Approach to the Examination Timetabling Problem". In: *Omega* 39.6, pp. 598 –607.

Turing, A. M. 1937. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of London Mathematical Society* 2.42, pp. 230–265.

Ülker, O., E. Özcan, and E. E. Korkmaz. 2007. "Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling". In: *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling VI*. PATAT'06. Springer-Verlag, pp. 347–363.

Vayssade, M. 1978. "Une approche informatique pour résoudre des problèmes de partitionnement complexes". PhD thesis. Université de Technologie de Compiègne.

Werra, D. de. 1985. "An introduction to timetabling". In: *European Journal of Operational Research* 19.2, pp. 151–162.

Wren, A. 1996. "Scheduling, timetabling and rostering—a special relationship?" In: *Practice and theory of automated timetabling*. Springer, pp. 46–75.